

September/
Oktober 1989
3. Jahrgang
Heft Nr. 5

DM 24,80
ÖS 175,-
sfr 24,80

84037/89005
ISSN 0933-9434

Microsoft System Journal

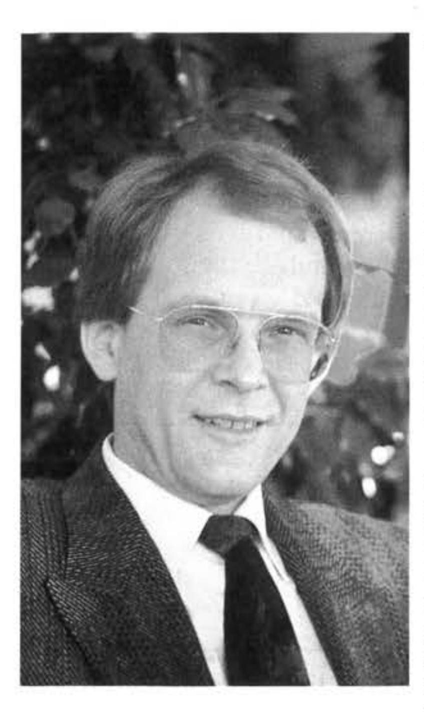
Hello! WOW Neon Block Fade Hello!

Basic • C • Pascal MS OS/2

GPI, die grafische Programmierschnittstelle des MS OS/2 Presentation Managers, kann Fonts nahezu genauso gut manipulieren wie PostScript. Der Windows- und MS OS/2-Experte Charles Petzold zeigt anhand vieler Beispiele, wie vielseitig und leistungsfähig die Vektorfonts unter dem MS OS/2-Presentation Managers sind.

MS OS/2
Vektorfonts unter dem Presentation Manager
Sprachen
Basic als professionelle Programmiersprache
QuickPascal und OOP
Virtuelle Speicherverwaltung in C
Windows
Hilfeverwaltung für Microsoft Windows
Hardware
Der Interrupt-Controller 8259A
Textverarbeitung
Das neue Microsoft Word 5.0
SAA-Serie
Dialogboxen in C
C-Kurs für Umsteiger
Teil 1: Die Grundlagen

Diskette in jedem Heft



Editorial

Dies ist bereits die zwölfte Ausgabe des *Microsoft System Journals*. Microsoft bietet damit seit nunmehr zwei Jahren deutschsprachigen Systementwicklern, Programmierern und professionellen Softwarenutzern alle zwei Monate eine Software-Zeitschrift als Ratgeber an.

Von Anfang an war es unsere Absicht, ein »Power-Blatt« für »Power-User« zu schaffen, ein Magazin, wie es im deutschen Zeitschriftenmarkt bislang nicht zu finden war. Mit Hilfe einer unabhängigen Redaktion ist es uns in den vergangenen beiden Jahren gelungen, das *Microsoft System Journal* als qualitativ hochwertige Software-Zeitschrift für Abonnenten zu etablieren.

Der Ruf des Blattes lebt von fachkundigen Redakteuren und freien Mitarbeitern, und nicht zuletzt vom Expertenpotential aus den Entwicklungsabteilungen von Microsoft. Die regelmäßigen Beiträge von renommierten Systementwicklern aus den USA, aus den Labors von Microsoft, aber auch anderer Softwareschmieden, haben das *System Journal* zu einem transatlantischen Knowhow-Importeur werden lassen.

Heute gibt es in Europa jedoch zahlreiche eigenständige Software-Entwicklungen. Unabhängige Programmierer haben ebenso wie große Computerhäuser qualitativ gegenüber dem »Mutterland« der PC-Technologie aufgeholt. Es ist deshalb an der Zeit, das *Microsoft System Journal* stärker als bisher für Originalbeiträge aus der Bundesrepublik Deutschland, der Schweiz und aus Österreich zu öffnen.

Zugleich wollen wir der Zeitschrift neue Leserkreise erschließen. Die vorliegende Ausgabe ist das erste Heft, das Microsoft in einer Kooperation mit dem Vogel Verlag auch über Kioske und den Fachbuchhandel vertreibt. So kommen Know-how und Erfahrung eines renommierten Verlages und das technologische Know-how von Microsoft und seiner Redaktion zum Nutzen einer breiteren Lesergemeinde zusammen.

Mit der Öffnung für Kioskvertrieb und Einzelverkauf wollen wir jedoch das technische Niveau nicht senken. Auch wenn eine Verbreiterung der Leserschaft mit einer thematischen Verbreiterung einhergeht, bleibt das *Microsoft System Journal* die Zeitschrift für die qualifizierte Minderheit der professionellen Entwickler. Wir werden uns nach wie vor der avanciertesten Technologien widmen. Im Mittelpunkt unserer Berichterstattung werden also auch künftig MS OS/2, Windows, SQL, Netzwerk- und Datenbanktechniken stehen. Wir werden vor allem über die Programmierung in modernen Hochsprachen wie C und FORTRAN berichten. Stärker als bisher werden wir auf aktuelle Tendenzen im High-End-Hardwaremarkt eingehen, also zum Beispiel auf EISA, MCA und Netzwerktechnik.

Unsere weiteren Themen aber beschließen Sie — unsere Leser! Ihnen gehört diese Zeitschrift, auf Ihre Bedürfnisse wollen wir eingehen, auf Ihre Erfahrungen sind wir angewiesen. Deshalb beachten Sie bitte unsere Leserbefragung auf Seite 101.

Und nun viel Spaß mit dem neuen *Microsoft System Journal*!

Christian Wedell

Geschäftsführer
der Microsoft GmbH

Inhalt

MS OS/2

Vektorfonts unter dem Presentation Manager — 6

GPI, die grafische Programmierschnittstelle des Presentation Managers, kann Fonts nahezu genauso gut manipulieren wie PostScript. Der Windows- und OS/2-Experte Charles Petzold zeigt, wie vielseitig und leistungsfähig die Vektorfonts des Presentation Managers sind.

Die Maus unter OS/2 — 23

Die Maus ist einfach zu programmieren und die Systemaufrufe von OS/2 für die Maus sind nicht schwieriger als die für die Tastatur, wie in diesem Buchauszug demonstriert wird.

Windows

Hilfe-Verwaltung für Windows — 30

Anwendungen mit einem Hilfe-Modul werden schneller erlernt, vielfältiger genutzt und letztendlich auch öfters verkauft. Jedes Programm sollte also ein Hilfe-Modul besitzen, auch Windows-Programme.

Sprachen

Basic als professionelle Programmiersprache — 59

Viele professionelle Programmierer denken, daß Basic eine Spielzeugsprache ist und nichts für ernsthafte Programmentwicklung. Diese Einschätzung beginnt sich zu ändern, wie Ethan Winer, der Autor einer bekannten Basic-Toolbox, in einem Interview erläutert.

Das neue QuickPascal von Microsoft — 68

Mit QuickPascal wird die überaus erfolgreiche Quick-Sprachen-Reihe um einen leistungsstarken Pascal-Compiler erweitert. Beeindruckend sind vor allem seine objektorientierten Leistungsmerkmale.

Microsoft C

Eine virtuelle Speicherverwaltung in C — 104

Unter MS-DOS ist der Hauptspeicher knapp, deshalb entstand die virtuelle Speicherverwaltung VMM (Virtual Memory Manager).

Erstellung und Verwaltung von SAA-Dialogboxen — 116

Dialogboxen kommt als Teil des SAA-Konzepts eine große Bedeutung zu, weshalb wir sie im Rahmen unserer SAA-Serie genauer unter die Lupe nehmen und eine Umsetzung in C zeigen.

Datenorganisation in C-Programmen
Strukturen sind einfach zu handhaben und unterstützen beispielhaft den korrekten Umgang mit Daten. Unions und typedef-Deklarationen sind schwieriger.
132

Hilfe für C-Entwickler
Rapid Prototyping-Sprachen ermöglicht eine nichtprozedurale Programmierung. Wir beschreiben dies am Beispiel des RAPID Prototyping-Systems.
141

Programmieren in C: Die Grundlagen
Ab dieser Ausgabe des Microsoft System Journals bringen wir in mehreren Folgen einen C-Kurs für Umsteiger, also für Leute, die bereits Erfahrungen mit anderen Programmiersprachen wie Pascal oder Basic gemacht haben.
144

Anwendungen

Dieses ist der fünfte Streich ...
... und der sechste, der kann ruhig noch ein wenig warten, denn mit der Version 5.0 – die ab September im Handel ist – hat Word wieder einen mächtigen Sprung nach vorn gemacht.
158

Hardware

Steuerung von Ein-/Ausgabegeräten
Hardwareinterrupts werden von einem Ein-/ Ausgabegerät initiiert und von dem Interruptcontroller 8259A gehandhabt.
166

Ein Scanner für alle Zwecke
Eine sinnvolle Ergänzung zu DTP- und Textverarbeitungssystemen stellt ein Scanner dar, mit dem bereits angefertigte Zeichnungen oder auch maschinengeschriebene Texte in den Computer gelangen.
174

Editorial **3**
Software **84**
Hardware **96**
Termine **98**
Leserumfrage **101**
Bücher **103**
Hotline **143**
Vorschau **178**
Impressum **178**
Inserentenverzeichnis **178**

Der Windows- und OS/2-Experte Charles Petzold zeigt, wie vielseitig und leistungsfähig die Vektorfonts des Presentation Managers sind.

Vektorfonts unter dem Presentation Manager

Lassen Sie mich mit einer Frage beginnen: Welche grafische Programmiersprache speichert Fonts als Linien und Kurven (und nicht als Bitmaps) und ermöglich es dadurch, daß die Fonts beliebig gestreckt, umrahmt, mit beliebigen Mustern gefüllt oder sogar als Clipping-Bereiche verwendet werden können? Eine naheliegende Antwort ist PostScript, die Seitenbeschreibungssprache von Adobe, die in vielen leistungsfähigen Laserdruckern verwendet wird (zum Beispiel dem Apple LaserWriter) und Fotosatzbelichtern wie den Linotronic-Systemen von Linotype. In den letzten Jahren ist PostScript zur Standardsprache für die Computermanipulation von Fonts und Text geworden. Eine gleichermaßen gültige Antwort ist GPI (Graphics Programming Interface), die grafische Programmierschnittstelle des OS/2 Presentation Managers.

In diesem Artikel wird gezeigt, wie man unter GPI mit Vektorfonts arbeitet und es werden viele PostScript-Techniken demonstriert. Wie Sie noch sehen werden, hat das GPI die Fähigkeiten, nahezu alles mit Fonts zu machen, was man auch unter PostScript tun kann. Das GPI hat jedoch einen kleinen Nachteil, auf den ich noch am Ende des Artikels eingehen werde.

Die Probleme mit Text

Die Anzeige von Text ist immer der problematischste Teil eines grafischen Programmiersystems. Anders als Linien und Polygone (die einfach nur mathematische Gebilde sind), richtet Text sich nach einer langen Tradition ästhetischer Typographie. Es muß in jedem Computergrafik-System das Ziel sein, Text ebenso sauber und leicht lesbar anzuzeigen, wie in einem gedruckten Buch. Doch die meisten Ausgabegeräte (wie Bildschirme und Drucker) sind digitale Medien. Die subtil geformten und gerundeten Zeichen, aus denen traditionelle Fonts bestehen, müssen für die Speicherung in einzelne Pixel zerlegt und auf dem Ausgabegerät wieder zusammengesetzt werden. Dies führt oft zu Verzerrungen in der Darstellung eines Textes.

Ein wesentlicher Vorteil der Verwendung eines Computers für diese Arbeit ist seine Vielseitigkeit. Man kann eine breite Palette von Fonts in verschiedenen Größen und Charakteristiken verwenden und sie für die Anzeige modifizieren. Das Ausmaß, in dem man Fonts verändern kann, hängt davon ab, wie Fonts gespeichert werden.

Bitmaps und Vektoren

Ein Font wird im allgemeinen im Computer (oder Drucker) auf eine von zwei Arten gespeichert. Zum einen kann ein Font als Abbild oder Bitmap gespeichert werden. Jedes Zeichen eines Fonts ist einfach nur ein rechteckiges Bit-Array. Null-Bits stellen dabei im allgemeinen den Hintergrund um das Zeichen herum dar und Einser-Bits bilden das Zeichen selbst. Zum anderen kann ein Font im Vektor- oder Umrißformat (Outline) gespeichert sein, wobei jedes Zeichen als eine Reihe von Linien und Kurven definiert ist, die Bereiche umschließen. Das Zeichen wird dadurch angezeigt, daß der Umriß auf dem Ausgabegerät gezeichnet und die umschlossenen Bereiche gefüllt werden.

Bitmap- und Vektorfonts haben jeweils eigene Vor- und Nachteile. Bitmap-Fonts werden immer in bestimmten Fontgrößen für bestimmte Geräteauflösungen erzeugt. Die Größe eines Bitmapfonts kann nicht auf einfache Weise geändert werden. (So führt beispielsweise die Vergrößerung eines Bitmapfonts durch Verdoppelung der Pixel-Zeilen und Spalten zumeist zu einer Ver-

◄◄ *Listing 1: VECTFONT*

```
#-----------------------
# VECTFONT make file
#-----------------------

CL=cl -c -G2sw -W3 $*.c

vectfont.obj : vectfont.c vectfont.h
     $(CL)

vf00.obj : vf00.c vectfont.h
     $(CL)

vf01.obj : vf01.c vectfont.h
     $(CL)

vf02.obj : vf02.c vectfont.h
     $(CL)

vf03.obj : vf03.c vectfont.h
     $(CL)

vf04.obj : vf04.c vectfont.h
     $(CL)

vf05.obj : vf05.c vectfont.h
     $(CL)

vf06.obj : vf06.c vectfont.h
     $(CL)

vf07.obj : vf07.c vectfont.h
     $(CL)

vf08.obj : vf08.c vectfont.h
     $(CL)

vf09.obj : vf09.c vectfont.h
     $(CL)

vf10.obj : vf10.c vectfont.h
     $(CL)

vf11.obj : vf11.c vectfont.h
     $(CL)

vf12.obj : vf12.c vectfont.h
     $(CL)

vf13.obj : vf13.c vectfont.h
     $(CL)

vf14.obj : vf14.c vectfont.h
     $(CL)

vf15.obj : vf15.c vectfont.h
     $(CL)

vectfont.res : vectfont.rc vectfont.h
     rc -r vectfont

vectfont.exe : vectfont.obj vf00.obj vf01.obj vf02.obj vf03.obj \
                            vf04.obj vf05.obj vf06.obj vf07.obj \
                            vf08.obj vf09.obj vf10.obj vf11.obj \
                            vf12.obj vf13.obj vf14.obj vf15.obj \
                            vectfont.def
     link @vectfont.lnk
     rc vectfont.res

vectfont.exe : vectfont.res
     rc vectfont.res
```

◄◄ *Listing 2: VECTFONT.LNK*

```
vectfont.obj +
vf00.obj + vf01.obj + vf02.obj + vf03.obj +
vf04.obj + vf05.obj + vf06.obj + vf07.obj +
vf08.obj + vf09.obj + vf10.obj + vf11.obj +
vf12.obj + vf13.obj + vf14.obj + vf15.obj
vectfont.exe /align:16
NUL
os2.lib
vectfont.def
```

gröberung der Zeichendarstellung.) Bitmapfonts können auch nur schlecht rotiert werden, außer eventuell um 90 Grad.

Vektorfonts sind wesentlich besser formbar. Da sie als eine Reihe von Linien und Kurven definiert sind, können Vektorfonts auf jede beliebige Größe gestreckt oder gestaucht und in jedem beliebigen Winkel gedreht werden. Vektorfonts sind nicht an eine bestimmte Geräteauflösung gebunden.

◄ *Listing 3: VECTFONT.DEF*

```
;-----------------------------------
; VECTFONT.DEF module definition file
;-----------------------------------

NAME           VECTFONT  WINDOWAPI

DESCRIPTION    'Vector Font Demo Program (C) Charles Petzold, 1988'
PROTMODE
HEAPSIZE       1024
STACKSIZE      8192
EXPORTS        ClientWndProc
```

◄ *Listing 4: VECTFONT.H*

```
/*------------------------
   VECTFONT.H header file
  ------------------------*/

#define ID_RESOURCE      1

#define IDM_NOTHING      0
#define IDM_24POINT      1
#define IDM_STRETCH      2
#define IDM_MIRROR       3
#define IDM_CHARANGLE    4
#define IDM_ROTATE       5
#define IDM_CHARSHEAR    6
#define IDM_SHADOW       7
#define IDM_HOLLOW       8
#define IDM_DROPSHADOW   9
#define IDM_BLOCK        10
#define IDM_NEON         11
#define IDM_FADE         12
#define IDM_SPOKES       13
#define IDM_WAVY         14
#define IDM_MODSPOKES    15

#define LCID_MYFONT 1L
#define ID_PATH     1L
#define PI          3.14159

LONG CreateVectorFont (HPS hps, LONG lcid, CHAR *szFacename) ;   // VF00
BOOL ScaleVectorFont (HPS hps, SHORT xPointSize, SHORT yPointSize) ;
BOOL ScaleFontToBox (HPS hps, LONG cbText, CHAR *szText, LONG cxBox,
                                                         LONG cyBox) ;
VOID QueryStartPointInTextBox (HPS hps, LONG cbText, CHAR *szText,
                                        POINTL *pptl) ;
VOID ColorClient (HPS hps, LONG cxClient, LONG cyClient, LONG lColor) ;

VOID Display_24Point (HPS hps, LONG cxClient, LONG cyClient) ;   // VF01
VOID Display_Stretch (HPS hps, LONG cxClient, LONG cyClient) ;   // VF02
VOID Display_Mirror (HPS hps, LONG cxClient, LONG cyClient) ;    // VF03
VOID Display_CharAngle (HPS hps, LONG cxClient, LONG cyClient) ; // VF04
VOID Display_Rotate (HPS hps, LONG cxClient, LONG cyClient) ;    // VF05
VOID Display_CharShear (HPS hps, LONG cxClient, LONG cyClient) ; // VF06
VOID Display_Shadow (HPS hps, LONG cxClient, LONG cyClient) ;    // VF07
VOID Display_Hollow (HPS hps, LONG cxClient, LONG cyClient) ;    // VF08
VOID Display_DropShadow (HPS hps, LONG cxClient, LONG cyClient) ;// VF09
VOID Display_Block (HPS hps, LONG cxClient, LONG cyClient) ;     // VF10
VOID Display_Neon (HPS hps, LONG cxClient, LONG cyClient) ;      // VF11
VOID Display_Fade (HPS hps, LONG cxClient, LONG cyClient) ;      // VF12
VOID Display_Spokes (HPS hps, LONG cxClient, LONG cyClient) ;    // VF13
VOID Display_Wavy (HPS hps, LONG cxClient, LONG cyClient) ;      // VF14
VOID Display_ModSpokes (HPS hps, LONG cxClient, LONG cyClient) ; // VF15
```

Im allgemeinen sind Bitmapfonts jedoch besser zu lesen als Vektorfonts. Es werden eine Reihe von Techniken verwendet, um Bitmapfonts so aufzubauen, daß das Auge sie für sauberer hält, als sie eigentlich sind. Vektorfonts können – besonders wenn sie auf Geräten mit geringer Auflösung angezeigt oder auf kleine Größen herunterskaliert werden – nur mit mathematischen Algorithmen angepaßt werden, was zur Zeit noch weniger befriedigend ist, als die Arbeit eines menschlichen Font-Designers. Ein weiterer Vorteil von Bitmapfonts ist die Performance, da Vektorfonts in der Regel für das Zeichnen jedes einzelnen Zeichens wesentlich mehr Zeit erfordern.

Die meisten üblichen Laserdrucker speichern Fonts als Bitmaps, entweder im Drucker selbst oder in Font-Kassetten. Der Drucker ist auf bestimmte Fontgrößen beschränkt und sie können nicht beliebig gedreht werden.

►► *Listing 5: VECTFONT.C*

```
/*----------------------------------------------
   VECTFONT.C -- Vector Font Demo Program
  ----------------------------------------------*/

#define INCL_WIN
#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

MRESULT EXPENTRY ClientWndProc (HWND, USHORT, MPARAM, MPARAM) ;

HAB hab ;

int main (void)
     {
     static CHAR  szClientClass [] = "VectFont" ;
     static ULONG flFrameFlags = FCF_TITLEBAR      | FCF_SYSMENU  |
                                 FCF_SIZEBORDER    | FCF_MINMAX   |
                                 FCF_SHELLPOSITION | FCF_TASKLIST |
                                 FCF_MENU ;
     HMQ          hmq ;
     HWND         hwndFrame, hwndClient ;
     QMSG         qmsg ;

     hab = WinInitialize (0) ;
     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass (hab, szClientClass, ClientWndProc,
                            CS_SIZEREDRAW, 0) ;

     hwndFrame = WinCreateStdWindow (HWND_DESKTOP, WS_VISIBLE,
                                     &flFrameFlags, szClientClass,
                                     " - Vector Font Demo", 0L,
                                     NULL, ID_RESOURCE, &hwndClient) ;

     WinSendMsg (hwndFrame, WM_SETICON,
                 WinQuerySysPointer (HWND_DESKTOP, SPTR_APPICON, FALSE),
                 NULL) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;

     WinDestroyWindow (hwndFrame) ;
     WinDestroyMsgQueue (hmq) ;
     WinTerminate (hab) ;
     return 0 ;
     }

MRESULT EXPENTRY ClientWndProc (HWND hwnd, USHORT msg, MPARAM mp1,
                                                       MPARAM mp2)
     {
     static struct {
                   SHORT idCmd ;
                   VOID (*fn) (HPS, LONG, LONG) ;
                   }
                   vectfont [] = {
                                 IDM_NOTHING,     NULL,
                                 IDM_24POINT,     Display_24Point,
                                 IDM_MIRROR,      Display_Mirror,
                                 IDM_STRETCH,     Display_Stretch,
                                 IDM_CHARANGLE,   Display_CharAngle,
                                 IDM_ROTATE,      Display_Rotate,
                                 IDM_CHARSHEAR,   Display_CharShear,
                                 IDM_SHADOW,      Display_Shadow,
                                 IDM_HOLLOW,      Display_Hollow,
                                 IDM_DROPSHADOW,  Display_DropShadow,
                                 IDM_BLOCK,       Display_Block,
                                 IDM_NEON,        Display_Neon,
                                 IDM_FADE,        Display_Fade,
                                 IDM_SPOKES,      Display_Spokes,
                                 IDM_WAVY,        Display_Wavy,
                                 IDM_MODSPOKES,   Display_ModSpokes
                                 } ;
     static HDC    hdc ;
     static HPS    hps ;
     static HWND   hwndMenu ;
     static POINTL ptlClient ;
     static SHORT  sNumRoutines = sizeof vectfont / sizeof vectfont[0],
                   sDisplay = IDM_NOTHING ;
     INT           i ;
     RECTL         rcl ;
     SIZEL         sizl ;

     switch (msg)
          {
          case WM_CREATE:
               hdc = WinOpenWindowDC (hwnd) ;
                                        // Create PS use Twips page units
               sizl.cx = 0 ;
               sizl.cy = 0 ;
               hps = GpiCreatePS (hab, hdc, &sizl,
                                  PU_TWIPS     | GPIF_DEFAULT |
                                  GPIT_MICRO   | GPIA_ASSOC) ;
                                       // Adjust Page Viewport for points
               GpiQueryPageViewport (hps, &rcl) ;
               rcl.xRight *= 20 ;
               rcl.yTop   *= 20 ;
               GpiSetPageViewport (hps, &rcl) ;

               hwndMenu = WinWindowFromID (
                               WinQueryWindow (hwnd, QW_PARENT, FALSE),
                               FID_MENU) ;
               return 0 ;

          case WM_SIZE:
               ptlClient.x = SHORT1FROMMP (mp2) ;      // client width
               ptlClient.y = SHORT2FROMMP (mp2) ;      // client height

               GpiConvert (hps, CVTC_DEVICE, CVTC_PAGE, 1L, &ptlClient);
               return 0 ;

          case WM_COMMAND:
               for (i = 0 ; i < sNumRoutines ; i++)
                    if (COMMANDMSG(&msg)->cmd == vectfont[i].idCmd)
                         {
                         if (sDisplay == COMMANDMSG(&msg)->cmd)
                              return 0 ;

                         WinSendMsg (hwndMenu, MM_SETITEMATTR,
                                     MPFROM2SHORT (sDisplay, TRUE),
                                     MPFROM2SHORT (MIA_CHECKED, 0)) ;

                         sDisplay = COMMANDMSG(&msg)->cmd ;

                         WinSendMsg (hwndMenu, MM_SETITEMATTR,
                                     MPFROM2SHORT (sDisplay, TRUE),
                                     MPFROM2SHORT (MIA_CHECKED,
                                                   MIA_CHECKED)) ;

                         WinInvalidateRect (hwnd, NULL, FALSE) ;
                         return 0 ;
                         }
               break ;

          case WM_PAINT:
               WinBeginPaint (hwnd, hps, NULL) ;
               GpiErase (hps) ;
                                        // Display hourglass pointer

               WinSetPointer (HWND_DESKTOP,
                    WinQuerySysPointer (HWND_DESKTOP,SPTR_WAIT,FALSE));

               if (!WinQuerySysValue (HWND_DESKTOP, SV_MOUSEPRESENT))
                    WinShowPointer (HWND_DESKTOP, TRUE) ;

                                        // Execute font routine

               for (i = 0 ; i < sNumRoutines ; i++)
                    if (sDisplay == vectfont[i].idCmd)
                         {
                         if (vectfont[i].fn != NULL)
                              {
                              GpiSavePS (hps) ;
                              vectfont[i].fn (hps, ptlClient.x,
                                                   ptlClient.y) ;
                              GpiRestorePS (hps, -1L) ;
                              }
                         break ;
                         }
                                        // Display arrow pointer

               if (!WinQuerySysValue (HWND_DESKTOP, SV_MOUSEPRESENT))
                    WinShowPointer (HWND_DESKTOP, FALSE) ;

               WinSetPointer (HWND_DESKTOP,
                    WinQuerySysPointer (HWND_DESKTOP,SPTR_ARROW,FALSE));

               WinEndPaint (hps) ;
               return 0 ;

          case WM_DESTROY:
               GpiDestroyPS (hps) ;
               return 0 ;
          }
     return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
     }
```

Wesentlich vielseitiger sind die Fonts, die in PostScript-Druckern gespeichert werden. Diese Fonts sind als Vektoren gespeichert. PostScript-Fonts können auf jede beliebige Größe vergrößert oder verkleinert werden, sie können beliebig gedreht, mit verschiedenen Mustern gefüllt und für das Clipping verwendet werden.

Die GPI-Fonts

Das GPI kann natürlich die Vorteile der Fonts nützen, die in Ausgabegeräten (z.B. Laserdrukkern) gespeichert und von ihnen unterstützt werden. Es enthält aber auch eigene Unterstützung für sowohl Bitmap- als auch Vektorfonts.

▶ *Tabelle 1: Die dynamischen Link-Library-Dateien von OS/2 1.1. Die Fontnamen stehen in Anführungszeichen.*

▶▶ *Listing 6: VFOO.C*

Dynamic-Link-Library-Datei Abbild-Fonts	Vector-Fonts
COURIER.FON "Courier" (8, 10 und 12 Punkte für CGA, EGA, VGA und IBM Proprinter)	"Courier" "Courier Bold" "Courier Italic" "Courier Bold Italic"
HELV.FON "Helv" (8, 10, 12, 14, 18 und 24 Punkte für CGA, EGA, VGA und IBM Proprinter)	"Helv" "Helv Bold" "Helv Italic" "Helv Bold Italic
TIMES.FON "Tms Rmn" (8, 10, 12, 14, 18 und 24 Punkte für CGA, EGA, VGA, und IBM Proprinter)	"Tms Rmn" "Tms Rmn Bold" "Tms Rmn Italic" "Tms Rmn Bold Italic"

Die Bitmapfonts waren zu erwarten, da sie sich besonders gut für Bildschirme mit geringer Auflösung und Matrixdrucker eignen. Bitmapfonts sind ein wichtiger Bestandteil der meisten grafischen Programmiersysteme (zum Beispiel des Microsoft Windows GDI).

Die Existenz von Vektorfonts im GPI ist eine wahre Wonne. GPI kann diese Vektorfonts mit jedem Ausgabegerät verwenden. So können nun verschiedene Fonttechniken, die bisher nur auf PostScript-Druckern verwendet werden konnten, auch auf anderen Laserdruckern und sogar auf dem Bildschirm eingesetzt werden.

Die OS/2 Version 1.1 wird mit drei dynamischen Link-Libraries geliefert, die nur Ressourcen enthalten und die Namenserweiterung FON tragen. Dabei handelt es sich um Fontdateien. Ihr Inhalt wird in der *Tabelle 1* beschrieben. Darüber hinaus können auch die Bildschirm- (DISPLAY.DLL) und Druckertreiber Fonts enthalten, die speziell für dieses Gerät entworfen wurden. So ist beispielsweise der proportionale Standardsystemfont in DISPLAY.DLL gespeichert.

Wenn Sie einen der Fonts in den Fontdateien verwenden wollen, müssen Sie sie über das Control Panel-Programm des Presentation Managers installieren. Man braucht nur einen Font aus jeder der Dateien installieren, und man braucht dies auch nur einmal zu tun.

Jeder Font hat einen Namen, der in der Tabelle 1 in Anführungszeichen steht. Jeder Bitmapfont ist in verschiedenen Punktgrößen und für verschiedene Ausgabegeräte verfügbar: CGA, EGA, VGA (und 8514/A) und den IBM Proprinter.

```
/*-----------------------------------------------------
   VFOO.C — Routines for working with vector fonts
  -----------------------------------------------------*/

#define INCL_WIN
#define INCL_GPI
#include <os2.h>
#include <stdlib.h>
#include <string.h>
#include "vectfont.h"

extern HAB hab ;

LONG CreateVectorFont (HPS hps, LONG lcid, CHAR *szFacename)
     {
     FATTRS fat ;

     fat.usRecordLength  = sizeof fat ;
     fat.fsSelection     = 0 ;
     fat.lMatch          = 0 ;
     fat.idRegistry      = 0 ;
     fat.usCodePage      = GpiQueryCp (hps) ;
     fat.lMaxBaselineExt = 0 ;
     fat.lAveCharWidth   = 0 ;
     fat.fsType          = 0 ;
     fat.fsFontUse       = FATTR_FONTUSE_OUTLINE |
                           FATTR_FONTUSE_TRANSFORMABLE ;

     strcpy (fat.szFacename, szFacename) ;

     return GpiCreateLogFont (hps, NULL, lcid, &fat) ;
     }

BOOL ScaleVectorFont (HPS hps, SHORT xPointSize, SHORT yPointSize)
     {
     HDC    hdc ;
     LONG   xDeviceRes, yDeviceRes ;
     POINTL ptlFont ;
     SIZEF  sizfx ;
                         // Get device resolution in pixels per meter

     hdc = GpiQueryDevice (hps) ;

     DevQueryCaps (hdc, CAPS_HORIZONTAL_RESOLUTION, 1L, &xDeviceRes) ;
     DevQueryCaps (hdc, CAPS_VERTICAL_RESOLUTION,   1L, &yDeviceRes) ;

                         // Find desired font size in pixels

     ptlFont.x = 254L * xPointSize * xDeviceRes / 7200000L ;
     ptlFont.y = 254L * yPointSize * yDeviceRes / 7200000L ;

                         // Convert to page units

     GpiConvert (hps, CVTC_DEVICE, CVTC_PAGE, 1L, &ptlFont) ;

                         // Set the character box

     sizfx.cx = MAKEFIXED (ptlFont.x, 0) ;
     sizfx.cy = MAKEFIXED (ptlFont.y, 0) ;

     return GpiSetCharBox (hps, &sizfx) ;
     }

BOOL ScaleFontToBox (HPS hps, LONG cbText, CHAR *szText, LONG cxBox,
                                                          LONG cyBox)
     {
     POINTL aptl[TXTBOX_COUNT] ;
     SIZEF  sizfx ;

     GpiQueryCharBox (hps, &sizfx) ;
     GpiQueryTextBox (hps, cbText, szText, TXTBOX_COUNT, aptl) ;

     sizfx.cx = sizfx.cx /
          (max (aptl[TXTBOX_TOPRIGHT].x, aptl[TXTBOX_BOTTOMRIGHT].x) -
           min (aptl[TXTBOX_TOPLEFT].x,  aptl[TXTBOX_BOTTOMLEFT].x))
          * cxBox ;

     sizfx.cy = sizfx.cy /
          (max (aptl[TXTBOX_TOPRIGHT].y,    aptl[TXTBOX_TOPLEFT].y) -
           min (aptl[TXTBOX_BOTTOMRIGHT].y, aptl[TXTBOX_BOTTOMLEFT].y))
          * cyBox ;

     return GpiSetCharBox (hps, &sizfx) ;
     }

VOID QueryStartPointInTextBox (HPS hps, LONG cbText, CHAR *szText,
                                        POINTL *pptl)
     {
     POINTL aptl[TXTBOX_COUNT] ;

     GpiQueryTextBox (hps, cbText, szText, TXTBOX_COUNT, aptl) ;

     pptl->x = max(-aptl[TXTBOX_TOPLEFT].x, -aptl[TXTBOX_BOTTOMLEFT].x);
     pptl->y = max(-aptl[TXTBOX_TOPLEFT].y, -aptl[TXTBOX_BOTTOMLEFT].y);
     }

VOID ColorClient (HPS hps, LONG cxClient, LONG cyClient, LONG lColor)
     {
     RECTL rcl ;

     WinSetRect (hab, &rcl, 0, 0, (SHORT) cxClient, (SHORT) cyClient) ;
     WinFillRect (hps, &rcl, lColor) ;
     }
```

Das GPI kann Variationen dieser Fonts wie kursive und fette Versionen erzeugen. Vektorfonts brauchen dagegen nicht für ein bestimmtes Ausgabegerät oder eine Punkgröße erzeugt werden, da sie beliebig skaliert werden können. Sie können der Tabelle entnehmen, daß auch kursive und fette Versionen der Fonts vorhanden sind.

Die Vektorfonts des GPI gleichen den Fonts Courier, Helvetica und Times, die in den meisten PostScript-Druckern enthalten sind.

Eine nähere Untersuchung der Fontdateien zeigt, daß die Vektorfonts als eine Reihe von GPI-Zeichenanweisungen codiert sind. Wenn es mit diesen Fonts Text zeichnet, setzt das GPI diese Zeichenanweisungen in GPI-Funktionen um, gewöhnlich GpiPolyLine zum Zeichnen von geraden Linien und GpiPolyFilletSharp zum Zeichnen von Kurven.

Das Programm VECTFONT

Das Programm VECTFONT demonstriert die Verwendung der GPI-Vektorfonts. Für eine bessere Übersicht habe ich das Programm in mehrere Module aufgeteilt. Die Dateien, die das Grundgerüst des Programms ausmachen, sind VECTFONT, VECTFONT.LNK, VECTFONT.DEF, VECTFONT.H und VECTFONT.C (*Listing 1 bis 5*).

Das Display-Menü von VECTFONT enthält 16 Optionen. Die erste Option (nichts anzeigen) ist die Standardeinstellung. Die anderen 15 Optionen entsprechen den Routinen in den Dateien VF01.C bis VF15.C, die weiter unten beschrieben werden (*Listings 8 bis 22*). Die Datei VF00.C (*Listing 7*) enthält einige Hilfsfunktionen, die von den Routinen in VF01.C bis VF15.C verwendet werden.

VECTFONT legt bei der Meldung WM_CREATE einen Micro-Präsentationsbereich an, wobei als Seiteneinheiten (page units) PU_TWIPS verwendet werden. (Twips ist ein Kunstwort, das ein Zwanzigstel eines Punkts bedeutet. Ein Punkt entspricht 1/72 eines Zolls, eine Seiteneinheit entspricht also 1/1440 eines Zolls.) VECTFONT modifiziert den rechteckigen Seiten-Viewport dann so, daß eine Seiteneinheit einem Punkt entspricht, was dem Standard-Koordinatensystem in PostScript entspricht.

Auch wenn VECTFONT auf dem Bildschirm ausgibt, sind Vektorfonts ganz offensichtlich besser für Laserdrucker geeignet. Wie Sie noch sehen werden, ist das Aussehen der Fonts (selbst bei 24 Punkt) nicht annähernd so gut, wie bei Bitmapfonts.

Sie werden auch feststellen, daß mehrere der Demonstrationsroutinen in VECTFONT einige Sekunden Laufzeit benötigen. Für andere Programme als Demonstrationsprogramme sollten Sie besser einen zweiten Ausführungsthread verwenden, um das Anhalten der Meldungsverarbeitung zu vermeiden.

Display

Courier - abcdefg
Courier Italic - abcdefg
Courier Bold - abcdefg
Courier Bold Italic - abcdefg
Tms Rmn - abcdefg
Tms Rmn Italic - abcdefg
Tms Rmn Bold - abcdefg
Tms Rmn Bold Italic - abcdefg
Helv - abcdefg
Helv Italic - abcdefg
Helv Bold - abcdefg
Helv Bold Italic - abcdefg

◄ *Bild 1: Die GPI-Vektorfonts in einer Größe von 24 Punkt*

◄ *Listing 7: VF01.C*

```
/*------------------------------------------------
   VF01.C — Display 24-point vector fonts
 ------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include <string.h>
#include "vectfont.h"

VOID Display_24Point (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR *szFacename[] = {
                                 "Courier",      "Courier Italic",
                                 "Courier Bold", "Courier Bold Italic",
                                 "Tms Rmn",      "Tms Rmn Italic",
                                 "Tms Rmn Bold", "Tms Rmn Bold Italic",
                                 "Helv",         "Helv Italic",
                                 "Helv Bold",    "Helv Bold Italic"
                                 } ;
     static INT  iNumFonts = sizeof szFacename / sizeof szFacename[0] ;
     FONTMETRICS fm ;
     INT         iFont ;
     POINTL      ptl ;

     ptl.x = cxClient / 8 ;
     ptl.y = cyClient ;

     for (iFont = 0 ; iFont < iNumFonts ; iFont++)
          {
                                   // Create font, select it and scale

          CreateVectorFont (hps, LCID_MYFONT, szFacename[iFont]) ;
          GpiSetCharSet (hps, LCID_MYFONT) ;
          ScaleVectorFont (hps, 240, 240) ;

                                   // Get font metrics for scaled font

          GpiQueryFontMetrics (hps, (LONG) sizeof (FONTMETRICS), &fm) ;
          ptl.y -= fm.lMaxBaselineExt ;

                                   // Display the font facename

          GpiCharStringAt (hps, &ptl, (LONG) strlen (szFacename[iFont]),
                           szFacename[iFont]) ;

          GpiCharString (hps, 10L, " - abcdefg") ;

          GpiSetCharSet (hps, LCID_DEFAULT) ;     // Clean up
          GpiDeleteSetId (hps, LCID_MYFONT) ;
          }
     }
```

Auswahl eines Outline-Fonts

Um in einem Presentation Manager-Programm einen Outline-Font zu verwenden, muß man zunächst einen logischen Font erzeugen und den Font dann in einen Präsentations-Bereich wählen. Die Funktion GpiCreateLogFont erzeugt einen logischen Font und verbindet ihn mit einer lokalen ID (man kann eine Zahl zwischen 1L und 254L wählen). Diese Funktion erfordert einen Zeiger auf eine Struktur vom Typ FATTRS (Fontattribute) die die Attribute des Fonts, den man möchte, angibt.

▸ *Bild 2: Ein Vektorfont in der Größe des Client-Fensters*

▸▸ *Bild 3: Vektorfont mit positiven/negativen Zeichenboxwerten*



▸ *Listing 8: VF02.C*

▸▸ *Listing 9: VF03.C*

```
/*------------------------------------------------------------
   VF02.C -- Display vector font stretched to client window
  ------------------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Stretch (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Hello!" ;
     static LONG cbText = sizeof szText - 1 ;
     POINTL      ptl ;

                              // Create font, select, and scale

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     GpiCharStringAt (hps, &ptl, cbText, szText) ;    // Display text

     GpiSetCharSet (hps, LCID_DEFAULT) ;               // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

```
/*------------------------------------------------------------
   VF03.C -- Display four strings in mirror reflections
  ------------------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Mirror (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Mirror" ;
     static LONG cbText = sizeof szText - 1 ;
     INT         i ;
     POINTL      ptl ;
     SIZEF       sizfx ;
                                   // Create font, select and scale

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient / 2, cyClient / 2) ;

     ptl.x = cxClient / 2 ;          // Center of client window
     ptl.y = cyClient / 2 ;

     for (i = 0 ; i < 4 ; i++)
          {
          GpiQueryCharBox (hps, &sizfx) ;

          if (i == 1 || i == 3)
               sizfx.cx *= -1 ;
                                   // Negate char box dimensions
          if (i == 2)
               sizfx.cy *= -1 ;

          GpiSetCharBox (hps, &sizfx) ;

          GpiCharStringAt (hps, &ptl, cbText, szText) ;
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;          // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;

     }
```

Um einen Vektorfont zu erzeugen, können die meisten Felder dieser Struktur auf Null gesetzt werden. Die wichtigsten Felder sind szFacename (das den Fontnamen enthält, einen der Namen in der letzten Spalte von Tabelle 1) und fsFontUse, das auf die konstanten Bezeichner FATTR_FONTUSE_OUTLINE und FATTR_FONTUSE_TRANSFORMABLE, kombiniert mit dem C-Bitoperator OR, gesetzt wird.

Eventuell ziehen Sie es vor, in VF00.C die Funktion CreateVectorFont zu verwenden. Diese Funktion erfordert nur eine Handle für den Präsentationsbereich, die lokale ID und den Fontnamen:

```
CreateVectorFont(hps, lcid, szFacename);
```

Nachdem sie einen logischen Font erzeugt haben (entweder mit GpiCreateLogFont oder CreateVectorFont), können Sie den Font in den Präsentationsbereich wählen:

```
GpiSetCharSet(hps, lcid);
```

Der Parameter lcid ist die lokale ID des Fonts. Nachdem der Font in den Präsentationsbereich gewählt wurde, kann man die Attribute des Fonts mit den verschiedenen weiter unten beschriebenen Funktionen verändern, Informationen über den Font mit GpiQueryFontMetrics, GpiQueryWidthTable und GpiQueryTextBox abfragen und den Font für die Textausgabe mit einer Textfunktion wie GpiCharStringAt verwenden.

Wenn Sie den Font nicht länger benötigen, wählen Sie einfach den Standardfont in den Präsentationsbereich:

```
GpiSetCharSet(hps, LCID_DEFAULT);
```

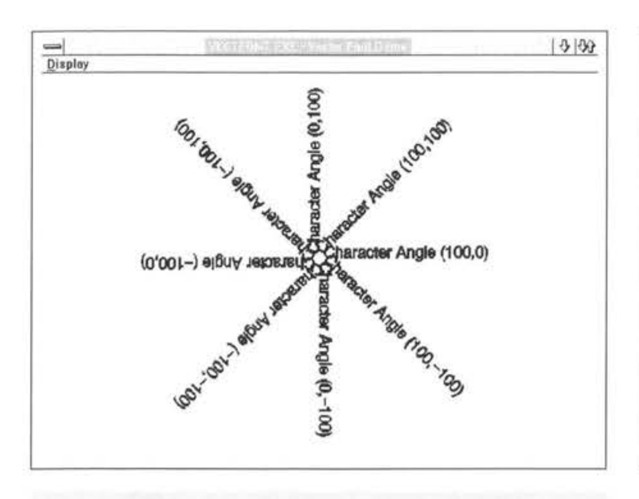
und löschen dann die lokale ID, die mit dem Font verknüpft wurde:

```
GpiDeleteSetId(hps, lcid);
```

In VECTFONT verwende ich immer den Bezeichner LCID_MYFONT für die lokale ID. Er wird in VECTFONT.H als 1L definiert.

Auf Punktgröße skalieren

Wenn Sie GpiSetCharSet aufrufen, um einen Vektorfont in den Präsentationsbereich zu wählen, hängen die ursprüngliche Breite und Höhe des Fonts von der GPI-Zeichenbox ab, die die Zeichenbreite und -höhe in Seiteneinheiten definiert.





◄◄ *Bild 4: Ein Vektorfont in acht Zeichenwinkeln*

◄ *Bild 5: Ein Zeichenstring auf einem Kreisumfang*

◄◄ *Listing 10: VF04.C*

◄ *Listing 11: VF05.C*

```
/*————————————————————————————————————
   VF04.C — Display eight character angles
  ————————————————————————————————————*/

#define INCL_GPI
#include <os2.h>
#include <stdio.h>
#include "vectfont.h"

VOID Display_CharAngle (HPS hps, LONG cxClient, LONG cyClient)
     {
     static GRADIENTL agradl[8] = { 100,    0,  100,  100,
                                      0,  100, -100,  100,
                                   -100,    0, -100, -100,
                                      0, -100,  100, -100 } ;
     CHAR             szBuffer[40] ;
     INT              iIndex ;
     POINTL           ptl ;

                                // Create Helvetica font

     CreateVectorFont (hps, LCID_MYFONT, "Helv") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleVectorFont (hps, 200, 200) ;

     ptl.x = cxClient / 2 ;   // Center of client window
     ptl.y = cyClient / 2 ;

     for (iIndex = 0 ; iIndex < 8 ; iIndex++)
          {
          GpiSetCharAngle (hps, agradl + iIndex) ;     // Char angle

          GpiCharStringAt (hps, &ptl,
               (LONG) sprintf (szBuffer, " Character Angle (%ld,%ld)",
                               agradl[iIndex].x, agradl[iIndex].y),
               szBuffer) ;
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;               // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

```
/*————————————————————————————————————
   VF05.C — Display "Hello, world" in circle
  ————————————————————————————————————*/

#define INCL_GPI
#include <os2.h>
#include <math.h>
#include <stdlib.h>
#include "vectfont.h"

VOID Display_Rotate (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Hello, world! " ;
     static LONG cbText = sizeof szText - 1L ;
     static LONG alWidthTable[256] ;
     double      ang, angCharWidth, angChar ;
     FONTMETRICS fm ;
     GRADIENTL   gradl ;
     INT         iChar ;
     LONG        lCircum, lRadius, lTotWidth, lCharRadius, cyChar ;
     POINTL      ptl ;

                         // Create the font and get font metrics

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;

     GpiQueryFontMetrics (hps, (LONG) sizeof fm, &fm) ;

                         // Find circle dimensions and scale font

     lRadius = min (cxClient / 4, cyClient / 4) ;
     lCircum = (LONG) (2 * PI * lRadius) ;
     cyChar  = fm.lMaxBaselineExt * lRadius / fm.lMaxAscender ;

     ScaleFontToBox (hps, cbText, szText, lCircum, cyChar) ;

                         // Obtain width table and total width

     GpiQueryWidthTable (hps, 0L, 256L, alWidthTable) ;

     for (lTotWidth = 0, iChar = 0 ; iChar < (INT) cbText ; iChar ++)
          lTotWidth += alWidthTable [szText [iChar]] ;

     ang = PI / 2 ;      // Initial angle for first character

     for (iChar = 0 ; iChar < (INT) cbText ; iChar++)
          {
                              // Set character angle

    angCharWidth = 2 * PI * alWidthTable [szText [iChar]] / lTotWidth ;

    gradl.x = (LONG) (lRadius * cos (ang - angCharWidth / 2 - PI / 2)) ;
    gradl.y = (LONG) (lRadius * sin (ang - angCharWidth / 2 - PI / 2)) ;

    GpiSetCharAngle (hps, &gradl) ;

                        // Find position for character and display it

          angChar = atan2 ((double) alWidthTable [szText [iChar]] / 2,
                           (double) lRadius) ;


          lCharRadius = (LONG) (lRadius / cos (angChar)) ;
          angChar += ang - angCharWidth / 2 ;

          ptl.x = (LONG) (cxClient / 2 + lCharRadius * cos (angChar)) ;
          ptl.y = (LONG) (cyClient / 2 + lCharRadius * sin (angChar)) ;

          GpiCharStringAt (hps, &ptl, 1L, szText + iChar) ;

          ang -= angCharWidth ;
          }

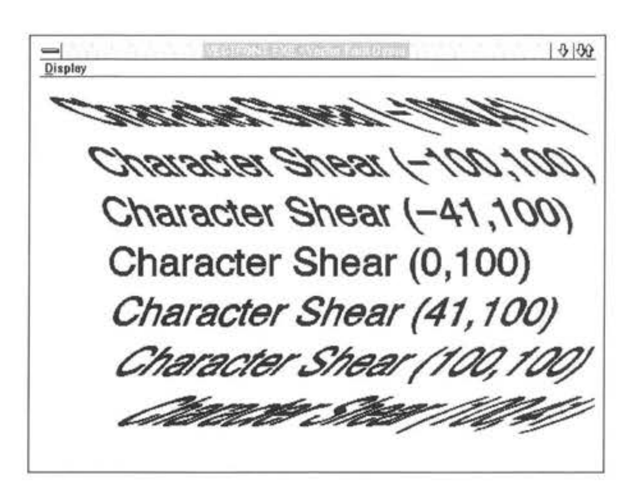
     GpiSetCharSet (hps, LCID_DEFAULT) ;          // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

Die Standard-Zeichenbox basiert auf der Größe des Standard-Systemfonts. Man kann die Größe der Zeichenbox durch einen Aufruf von GpiSetCharBox verändern.

Auf einen wichtigen Punkt muß hier hingewiesen werden: Um einen richtig proportionierten Vektorfont zu erhalten, muß man die Zeichenboxgröße verändern. Verwenden Sie nicht die Standardeinstellung. Im allgemeinen stellen Sie die Höhe einer Zeichenbox auf die gewünschte Höhe des Fonts ein. Wenn Sie möchten, daß ein Vektorfont eine normale Breite hat, stellen Sie die Breite der Zeichenbox auf den gleichen Wert, wie die Höhe. Für einen schmaleren Font stellen Sie die Breite der Zeichenbox kleiner als die Höhe ein, für einen breiteren Font größer als die Höhe.

Wenn Sie mit der Seiteneinheit PU_PELS arbeiten, müssen Sie die Ausmaße der Zeichenbox auch noch an die Unterschiede in der waagrechten und senkrechten Auflösung des Ausgabe-

▶ *Bild 6: Dies ist kein Bildschirmfehler, sondern ein Beispiel für die Zeichenneigung*



▶ *Listing 12: VF06.C*

```
/*------------------------------------------------------------------
   VF06.C — Display seven different character shear angles
  ------------------------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include <stdio.h>
#include "vectfont.h"

VOID Display_CharShear (HPS hps, LONG cxClient, LONG cyClient)
     {
     static POINTL aptlShear[7] = { -100,  41, -100, 100,
                                     -41, 100,    0, 100,
                                      41, 100,  100, 100,
                                     100,  41 } ;
     CHAR          szBuffer[40] ;
     FONTMETRICS   fm ;
     INT           iIndex ;
     POINTL        ptl ;

                              // Create and scale Helvetica font

     CreateVectorFont (hps, LCID_MYFONT, "Helv") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleVectorFont (hps, 480, 480) ;

                              // Get font metrics for scaled font

     GpiQueryFontMetrics (hps, (LONG) sizeof (FONTMETRICS), &fm) ;

     ptl.x = cxClient / 8 ;
     ptl.y = cyClient ;

     for (iIndex = 0 ; iIndex < 7 ; iIndex++)
          {
          GpiSetCharShear (hps, aptlShear + iIndex) ;  // Char shear

          ptl.y -= fm.lMaxBaselineExt ;

          GpiCharStringAt (hps, &ptl,
               (LONG) sprintf (szBuffer, "Character Shear (%ld,%ld)",
                          aptlShear[iIndex].x, aptlShear[iIndex].y),
               szBuffer) ;
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;                 // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

geräts anpassen. Aus diesem Grund ist es wesentlich einfacher, bei Vektorfonts eine der metrischen Seiteneinheiten (PU_LOENGLISH, PU_HIENGLISH, PU_LOMETRIC, PU_HIMETRIC oder PU_TWIPS) zu verwenden. Bei diesen Seiteneinheiten sind die waagrechten und senkrechten Seiteneinheiten gleich. Nehmen wir zum Beispiel an, sie verwenden die Seiteneinheit PU_TWIPS. Dies bedeutet, daß eine Seiteneinheit 1/20 eines Punktes oder 1/1440 eines Zolls entspricht. Nach der Wahl eines Vektorfonts in den Präsentationsbereich möchten Sie den Font auf 24 Punkt skalieren. Zunächst definieren Sie dazu eine Struktur vom Typ SIZEF:

```
SIZEF sizfx;
```

Die beiden Felder dieser Struktur mit den Namen cx und cy werden als 32-Bit-FIXED-Zahlen interpretiert, das bedeutet, die höherwertigen 16 Bit werden als Integer und die niederwertigen 16 Bit als Bruch interpretiert.

Wenn Sie einen Vektorfont auf eine 24-Punkt-Höhe skalieren wollen, können Sie das Makro MAKEFIXED verwenden, um die Felder der Struktur folgendermaßen zu belegen:

```
sizfx.cx = MAKEFIXED(24 * 20, 0);
sizfx.cy = MAKEFIXED(24 * 20, 0);
```

Die Multiplikation mit 20 ist notwendig, um die Punktgröße in Twips umzuwandeln. Rufen Sie dann GpiSetCharBox auf:

```
GpiSetCharBox(hps, &sizfx);
```

nach der Einstellung der Zeichenbox spiegelt jedes Zeichen- oder Textmaß, das Sie mit GpiQueryFontMetrics, GpiQueryTextBox und GpiQueryWidthTable abfragen, die neue Fontgröße wider.

Die Routine ScalevectorFont in VF00.C kann bei der Skalierung eines Vektorfonts auf die gewünschte Fontgröße behilflich sein. Diese Funktion arbeitet mit beliebigen Seiteneinheiten, sogar mit PU_PELS. Der zweite und dritte Parameter von ScaleVectorFont geben eine Punktgröße in einer Einheit von 0,1 Punkt an. (240 steht zum Beispiel für 24 Punkt.) Wenn Sie einen normal proportionierten Vektorfont wünschen, setzen Sie den dritten Parameter auf den gleichen Wert wie den zweiten.

Die Datei VF01.C in *Listing 8* zeigt, wie die bisher besprochenen Funktionen dazu verwendet werden können, um alle verfügbaren Vektorfonts in einer Größe von 24 Punkt anzuzeigen. Sie können die Funktion aus VF01.C starten, indem Sie die Option »24 Point Fonts« aus dem »Display«-Menü von VECTFONT auswählen. Der Quellcode ist in *Listing 8* zu sehen, sein Ergebnis in *Bild 1*.

Beliebige Größen

Neben der Skalierung des Vektorfonts auf eine bestimmte Punktgröße kann man Vektorfonts auch so skalieren, daß Sie in ein beliebiges Rechteck passen. Als Beispiel sei genannt, daß man einen kurzen Textstring so skalieren möchte, daß er das Client-Fenster ausfüllt.

Die Funktion in VF02.C (*Listing 9*) zeigt, wie dies gemacht wird. Sie können diese Funktion mit dem Menüpunkt »Stretched Font« aus dem Menü von VECTFONT starten. Die Funktion zeigt das Wort »Hello!« in der Schrift Tms Rmn Italic und in der Größe des Client-Fensters an.

Die Funktion ScaleToFontBox in VF00.C hilft bei der Erledigung dieses Jobs. Diese Funktion ruft zunächst GpiQueryTextBox auf, um die Koordinaten des Parallelogramms zu erhalten, das den Textstring umgibt. Die Zeichenbox wird dann auf der Basis der Größe dieser Textbox und des Rechtecks, in das der Text eingepaßt werden

soll, skaliert. Die Funktion QueryStartPointInTextBox in VF00.C bestimmt den Startpunkt des Textstrings (das heißt, den Punkt auf der Grundlinie links vom ersten Zeichen) innerhalb dieses Rechtecks.

Gespiegelte Abbilder

Mit der Zeichenbox kann man außer der Skalierung des Fonts auf eine bestimmte Größe auch die Spiegelung von Zeichen an einer waagrechten oder senkrechten Achse erzielen. Wenn die Höhe der Zeichenbox (das Feld cy der Struktur SIZEF) negativ ist, werden die Zeichen an der Grundlinie gespiegelt und auf dem Kopf stehend ausgegeben. Wenn die Breite der Zeichenbox (das Feld cx) negativ ist, werden die einzelnen Zeichen an der senkrechten Achse gespiegelt. Darüber hinaus zeichnet GpiCharStringAt ein Zeichen dann von rechts nach links. Das ist so, als würde der ganze Zeichenstring an der senkrechten Linie an der linken Seite des ersten Zeichens gespiegelt.

Die Funktion in VF03.C (*Listing 10*) zeigt den gleichen String viermal an und verwendet dabei alle möglichen Kombinationen negativer und positiver Zeichenboxwerte. Sie können diese Funktion mit dem Menüpunkt »Mirrored Font« auswählen.

Transformationen

Anders als Bitmapfonts können Vektorfonts auf jede beliebige Größe skaliert und in jedem beliebigen Winkel geneigt oder gedreht werden. Dies kann mit Matrixtransformationen erzielt werden. Darüber hinaus unterstützt das GPI auch mehrere spezielle Funktionen für die Transformation von Vektorfonts. Es wurde bereits gezeigt, wie die Funktion GpiSetCharBox die Skalierung von Fontzeichen ermöglicht. GpiSetCharAngle dreht die Fontzeichen und GpiSetCharShear neigt die Zeichen.

Zeichenwinkel und Drehung

Standardmäßig liegt die Grundlinie eines Vektorfontzeichens parallel zur X-Achse der Weltkoordinaten. Man kann dies mit GpiSetCharAngle ändern. Dadurch werden die Zeichen des Vektorfonts gedreht.

Der Zeichenwinkel wird mit der Struktur GRADIENTL angegeben, die aus zwei Feldern mit den Namen x und y vom Typ LONG besteht. Stellen Sie sich eine Linie vom Punkt (0,0) bis zum Punkt (x,y) in Weltkoordinaten vor. Die Grundlinie jedes Zeichens liegt parallel zu dieser Linie. Die Richtung des Textes entspricht der Richtung von (0,0) nach (x,y).



◄ *Bild 7: Zeichenstrings mit Schatten durch Zeichenneigung*

◄ *Listing 13: VF07.C*

```
/*-----------------------------------------------------
   VF07.C -- Display characters with sheared shadow
  -----------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Shadow (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Shadow" ;
     static LONG cbText = sizeof szText - 1 ;
     POINTL      ptl, ptlShear ;
     SIZEF       sizfx ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, 3 * cxClient / 4, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     ColorClient (hps, cxClient, cyClient, CLR_BLUE) ;

     GpiSavePS (hps) ;
     ptlShear.x = 200 ;                                   // Set char shear
     ptlShear.y = 100 ;
     GpiSetCharShear (hps, &ptlShear) ;

     GpiQueryCharBox (hps, &sizfx) ;
     sizfx.cy += sizfx.cy / 4 ;                          // Set char box
     GpiSetCharBox (hps, &sizfx) ;

     GpiSetColor (hps, CLR_DARKBLUE) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;       // Display shadow

     GpiRestorePS (hps, -1L) ;
     GpiSetColor (hps, CLR_RED) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;       // Display text

     GpiSetCharSet (hps, LCID_DEFAULT) ;                 // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

Sie können sich dies auch in trigonometischen Begriffen verdeutlichen. Die Grundlinie des Textes liegt parallel zu einer Linie mit dem Winkel a, gegen den Uhrzeigersinn von der X-Achse aus gemessen wobei

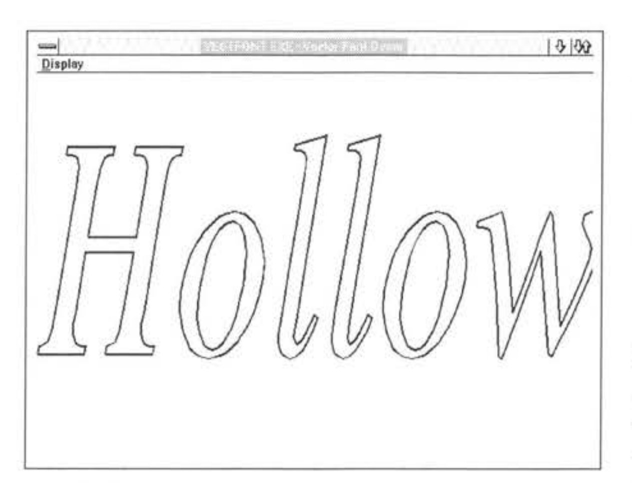
`a = arctan (y/x)`

ist, und y und x die beiden Felder der Struktur GRADIENTL sind.

Die absolute Größe von x und y sind unwichtig. Wichtig sind die relativen Größen und die Vorzeichen. Die Vorzeichen von x und y bestimmen die Richtung des Textstrings wie das der folgenden Tabelle zu entnehmen ist:

x	y	Richtung
+	+	nach oben rechts
–	+	nach oben links
–	–	nach unten links
+	–	nach unten rechts

► *Bild 8: Ausgehöhlte Zeichen*



► *Listing 14: VF08.C*

```
/*————————————————
   VF08.C — Hollow font
  ————————————————*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Hollow (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Hollow" ;
     static LONG cbText = sizeof szText - 1 ;
     POINTL      ptl ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     GpiBeginPath (hps, ID_PATH) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Text in path
     GpiEndPath (hps) ;

     GpiStrokePath (hps, ID_PATH, 0L) ;                // Stroke path

     GpiSetCharSet (hps, LCID_DEFAULT) ;               // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

Die Funktion in VF04.C (*Listing 11*) verwendet die Funktion GpiSetCharAngle um acht Textstrings in 45-Grad-Schritten um den Mittelpunkt des Client-Fensters herum anzuzeigen. Bei jedem String werden auch die Werte der Struktur GRADIENTL, die für den jeweiligen Aufruf verwendet wurden, mit ausgegeben.

In diesem Beispiel beginnt der Textstring mit einem Leerzeichen, um ein Durcheinander durch Überlappen in der Mitte des Client-Fensters zu vermeiden. Der Zeichenwinkel beeinflußt die Interpretation der Startposition des bei der Funktion GpiCharStringAt angegeben Strings nicht. Wenn Sie Ihren Kopf so drehen, daß ein String von links nach rechts läuft, bezeichnet die Startposition immer noch den Punkt auf der Grundlinie links vom ersten Zeichen.

Man kann die Zeichen eines Textes an einer gekrümmten Linie ausrichten, indem man den Startpositionwinkel jedes einzelnen Zeichens einzeln berechnet und die Zeichen auch einzeln anzeigt. Dies wird in VF05.C gemacht (*Listing 12*), um »Hello, World!« auf einer Kreislinie anzuzeigen.

Der Textstring wird abhängig vom Kreisumfang eines Kreises, der einen Radius von der Hälfte der Höhe bzw. Breite (der kleinere Wert) des Client-Fensters hat, im Mittelpunkt des Fensters skaliert. Die Funktion GpiQueryWidthTable wird dazu verwendet, die Breite der einzelnen Zeichen abzufragen und sie um den Kreis herum zu verteilen.

Zeichenneigung

Man kann Zeichenwinkel leicht mit Zeichenneigung verwechseln, deshalb eine kurze Beschreibung der Unterschiede. Der Zeichenwinkel bezieht sich auf die Orientierung der Grundlinie. Wie in den *Bildern 4* und *5* zu sehen ist, wird Text mit verschiedenen Zeichenwinkeln gedreht, jedoch ansonsten nicht verzerrt.

Die Zeichenneigung beeinflußt die Erscheinungsweise der Zeichen selbst, ganz unabhängig von einer Drehung. Bei der Zeichenneigung werden Zeichen nach links oder rechts geneigt, die untere Seite jedes Zeichens bleibt aber parallel zur X-Achse. Man kann die Zeichenneigung dazu verwenden, kursive Versionen eines Fonts zu erzeugen.

Um die Zeichenneigung einzustellen, ruft man die Funktion GpiSetCharShear auf. Dies Funktion benötigt einen Zeiger auf eine Struktur vom Typ POINTL, die aus zwei Feldern mit den Namen x und y besteht. Stellen Sie sich eine Linie vor, die von (0,0) bis (x,y) in Weltkoordinaten gezeichnet wird. Die linke und rechte Seite eines Zeichen verlaufen parallel zu dieser Linie.

Die Funktion in VF06.C (*Listing 13*) zeigt sieben Textstrings mit unterschiedlichen Zeichenneigungen an. Sie können diese Funktion starten, indem Sie den Menüpunkt »Character Shear« wählen. Zu jedem String werden die X- und Y-Werte der POINTL-Struktur, die die Zeichenneigung angibt, ausgegeben.

Die Zeichenneigung wird von der relativen Größe und den Vorzeichen der X- und Y-Werte der POINTL-Struktur bestimmt. Wenn das Vorzeichen beider Felder gleich ist, neigen sich die Zeichen nach rechts; wenn sie verschieden sind, neigen sich die Zeichen nach links. Die Zeichenneigung stellt die Zeichen nicht auf den Kopf. So hat eine Zeichenneigung mit dem Punkt (100, 100) den gleichen Effekt wie die mit (-100,-100).

Der Winkel der rechten und linken Seite der Zeichen zur Y-Achse wird manchmal der Neigungswinkel genannt. Theoretisch kann der Neigungswinkel bis etwas über –180 Grad (unendliche linke Neigung) und etwas unter +180 Grad (unendliche rechte Neigung) reichen; er ist identisch mit

```
a = arctan (x/y)
```

wobei x und y die beiden Felder der POINTL-Struktur sind.

Wenn man eine nicht standardmäßige Zeichenneigung einstellt, übergibt die Funktion GpiQueryTextBox ein Array von Punkten, die ein Parallelogramm (und kein Rechteck mehr) definieren. Die Breite der oberen und unteren Seite

dieser Textbox sind identisch mit der des nicht geneigten Textstrings und die Entfernung zwischen der oberen und unteren Seite bleibt auch gleich.

Man kann die Zeichenneigung dazu verwenden, um einen kursiven Schatten eines Textstrings zu zeichnen. Die Funktion VF07.C (*Listing 14*) färbt den Hinmtergrund des Fensters blau und zeigt den Textstring »Shadow« zweimal an. Der erste Aufruf von GpiCharStringAt zeigt den Schatten an. Dieser wird in dunklem Blau mit positiver Zeichenneigung gezeichnet. Der zweite Aufruf von GpiCharStringAt zeigt die Zeichen aufrecht in Rot mit einer etwas geringeren Zeichenboxhöhe an. Sie können diese Funktion starten, indem Sie den Menüpunkt »Font with Shadow« wählen.

Eine Kurzeinführung in Pfade

Um weitere Möglichkeiten der Vektorfonts zu untersuchen, ist es notwendig, uns mit GPI-Pfaden vertraut zu machen, die im wesentlichen den PostScript-Pfaden entsprechen. Im GPI legt man einen Pfad an, indem man zwischen Aufrufen der Funktionen GpiBeginPath und GpiEndPath Funktionen zum Zeichnen von Linien aufruft:

```
GpiBeginPath(hps, idPath);
        <... Linienfunktionen aufrufen ...>
GpiEndPath
```

Dies wird Pfadklammer genannt. In OS/2 1.1 muß idPath auf 1L gesetzt werden. Die Funktionen, die innerhalb einer Pfadklammer gültig sind, sind in der Dokumentation der Presentation Manager-Funktionen aufgeführt.

Die Funktionen, die man innerhalb einer Pfadklammer aufruft zeichnen nichts. Statt dessen merkt sich das System den Pfad, der durch diese Linien dargestellt wird. Oft umschließen die Linien, die man als Pfad zeichnet, Bereiche, doch das muß nicht sein.

Nach dem Aufruf von GpiEndPath kann man eines von drei Dingen mit dem so erzeugten Pfad anstellen:

• Man kann GpiStrokePath aufrufen, um die Linien zu zeichnen, die den Pfad ausmachen. Diese Linien werden mit der geometrischen Linienbreite, den Linienverbindungen und -enden gezeichnet (Beschreibung folgt später).

• Man kann GpiFillPath aufrufen, um die von dem definierten Pfad eingeschlossenen Bereiche zu füllen. Alle offenen Bereiche werden automatisch geschlossen. Der Bereich wird mit dem aktuellen Muster gefüllt.

• Man kann GpiSetClipPath aufrufen, um die vom Pfad eingeschlossenen Bereiche zu einem Clipping-Bereich zu machen. Alle offenen Bereiche werden automatisch geschlossen. Nachfolgende GPI-Aufrufe erzeugen ihre Ausgabe nur innerhalb der durch den Pfad beschriebenen Bereiche.



◂ *Bild 9: Zeichen mit hinterlegtem Schatten*

◂ *Listing 15: VF09.C*

```
/*————————————————————————————
   VF09.C — Font with Drop Shadow
  ————————————————————————————*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_DropShadow (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Hello!" ;
     static LONG cbText = sizeof szText - 1 ;
     POINTL      ptl ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Shadow

     ptl.x -= 12 ;        // 1/6 inch
     ptl.y += 12 ;

     GpiSetColor (hps, CLR_BACKGROUND) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Text string

     GpiBeginPath (hps, ID_PATH) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Outline
     GpiEndPath (hps) ;

     GpiSetColor (hps, CLR_NEUTRAL) ;
     GpiStrokePath (hps, ID_PATH, 0L) ;

     GpiSetCharSet (hps, LCID_DEFAULT) ;               // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

Jede dieser Funktionen bewirkt, daß der Pfad gelöscht wird. Vor dem Aufruf einer dieser drei Funktionen kann man noch die Funktion GpiModifyPath aufrufen, die ich gegen Ende dieses Artikels beschreiben werde.

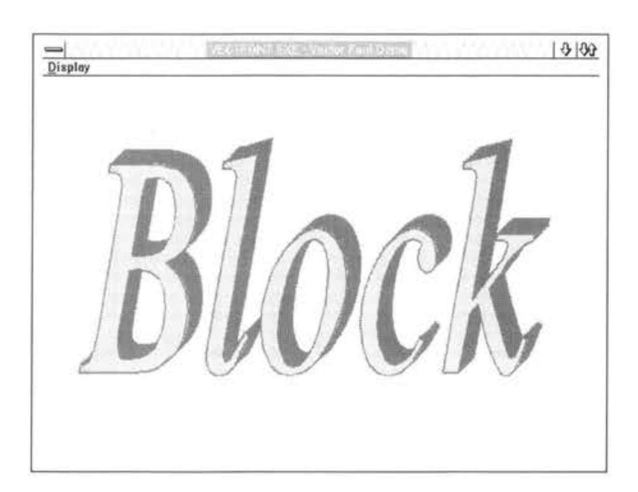
Normalerweise sind GpiCharStringAt und die anderen Textausgabefunktionen innerhalb einer Pfadklammer nicht gültig. Die einzige Ausnahme besteht dann, wenn ein Vektorfont in den Präsentationsbereich gewählt wird. GpiCharStringAt zeichnet den Textstring dann nicht. Statt dessen werden dann die Umrisse des Pfads Bestandteil des Pfads.

Die Pfade eröffnen eine ganze Sammlung von PostScript-ähnlichen Techniken, die man mit Vektorfonts verwenden kann.

Hohle Zeichen

Wir wollen damit beginnen, daß wir GpiCharStringAt in einer Pfadklammer aufrufen und dann GpiStrokePath verwenden, um die Linien

▶ *Bild 10: Massive Blockzeichen*



▶ *Listing 16: VF10.C*

```
/*-----------------------------
   VF10.C — Solid block font
  -----------------------------*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Block (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = " Block " ;
     static LONG cbText = sizeof szText - 1 ;
     INT         i ;
     POINTL      ptl ;


     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     ColorClient (hps, cxClient, cyClient, CLR_WHITE) ;
     GpiSetColor (hps, CLR_DARKGREEN) ;

     for (i = 0 ; i < 18 ; i++)
          {
          GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Block

          ptl.x -= 1 ;
          ptl.y -= 1 ;
          }

     GpiSetColor (hps, CLR_GREEN) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Text string

     GpiBeginPath (hps, ID_PATH) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Outline
     GpiEndPath (hps) ;

     GpiSetColor (hps, CLR_DARKGREEN) ;
     GpiStrokePath (hps, ID_PATH, 0L) ;

     GpiSetCharSet (hps, LCID_DEFAULT) ;               // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

des Pfads zu zeichnen. GpiStrokeAt hat die folgende Syntax:

`GpiStrokePath(hps, idPath, 0L);`

In der ersten Version des Presentation Managers muß der letzte Parameter der Funktion auf 0L gesetzt werden. Wenn man die Funktion verwendet, um einen mit GpiCharStringAt erzeugten Pfad zu zeichnen, werden nur die Umrisse gezeichnet und nicht das Innere. Dadurch werden hohle Zeichen erzeugt.

Die Funktion VF08.C (*Listing 15*) verwendet diese Technik, um die Umrisse der Zeichen des Textstrings »Hollow« zu zeichnen. Sie können diese Funktion mit dem Menüpunkt »Hollow Font« starten.

Vielleicht möchten Sie die Zeichen in einer Farbe und die Umrisse in einer anderen anzeigen. In diesem Fall müssen Sie GpiCharStringAt zweimal aufrufen, zunächst um das Innere in einer bestimmten Farbe zu zeichnen, und dann innerhalb einer Pfadklammer gefolgt von einem GpiStrokePath-Aufruf, um den Umriß zu zeichnen. Die Funktion in VF09.C (*Listing 16*) macht etwas ähnliches, um Text mit einem hinterlegten Schatten zu zeichnen.

Der Schatten wird zunächt mit einem normalen GpiCharStringAt-Aufruf gezeichnet. Eine weitere GpiCharStringAt-Funktion zeichnet den Text dann noch einmal in der aktuellen Fensterhintergrundfarbe 1/6 Zoll gegenüber dem ersten String versetzt. Dieser Text wiederum wird mit einem Umriß umgeben, der mit einem dritten GpiCharStringAt-Aufruf innerhalb einer Pfadklammer und gefolgt von GpiStrokePath erzeugt wird.

Ganz ähnlich verläuft die Erzeugung von Zeichen, die wie massive Blöcke aussehen, wie das mit der Funktion in VF10.C gemacht wird (*Listing 17*). Es werden 18 Zeichenstrings jeweils um einen Punkt versetzt in der Farbe CLR_DARKGREEN gezeichnet. Darüber wird der Zeichenstring noch einmal in CLR_GREEN gezeichnet und der Rand in CLR_DARKGREEN.

Den Pfad füllen

Bei der Vorstellung der Pfade habe ich erwähnt, daß man eines von drei Dingen mit Pfaden machen kann: zeichnen, füllen oder für das Clipping verwenden. Lassen Sie uns zum zweiten übergehen, zum Füllen des Pfads. Um einen Pfad zu füllen ruft man diese Funktion auf:

`GpiFillPath(hps, idPath, lOption);`

Dadurch wird der Pfad mit dem aktuellen Muster gefüllt. Alle offenen Bereiche werden vor dem Füllen automatisch geschlossen. Der Parameter lOption kann entweder FPATH_ALTERNATE oder FPATH_WINDING sein, um den Pfad im alternativen oder im Winding-Modus zu zeichnen. Bei Vektorfonts bewirkt FPATH-WINDING, daß das Innere einiger Buchstaben, zum Beispiel des O, gefüllt wird; Sie werden deshalb wahrscheinlich FPATH_ALTERNATE verwenden wollen.

Wenn das aktuelle Füllmuster für Bereiche PATSYM_ SOLID ist, bewirken folgende Aufrufe

```
GpiBeginPath(hps, idPath);
GpiCharStringAt(hps, &ptl, cch, szText);
GpiEndPath(hps);
GpiFillPath(hps, idPath, FPATH_ALTERNATE);
```

bei einem Vektorfont nahezu dasselbe, wie GpiCharStringAt selbst. Bei der Verwendung wird man das Muster auf einen anderen Wert als PATSYM_SOLID einstellen. (Ein Fehler in OS/2 1.1 bewirkt, daß das aktuelle Muster bei einer Pfadklammer, in der GpiCharStringAt aufgerufen wird, auf PATSYM_SOLID zurückgesetzt wird. Man kann diesen Fehler umgehen, indem man nach Beendigung des Pfads GpiSetPattern aufruft.)

Die Funktion in VF11.C (*Listing 18*) verwendet GpiFillPath, um den Textstring »Fade« achtmal hintereinander mit den acht GPI-Schraffurmustern (PATSYM_DENSE1 bis PATSYM_-DENSE8) zu zeichnen. Schließlich wird dann noch GpiCharStringAt einmal außerhalb der Pfadklammer aufgerufen. Sie können diese Funktion mit dem Menüpunkt »Fading Font« aufrufen.

Geometrisch dicke Linien

Auf den ersten Blick sieht es so aus, als gäbe es keinen Unterschied zwischen dem Zeichnen einer Linie in dieser Weise:

```
GpiMove(hps, &ptlBeg);
GpiLine(hps, &ptlEnd);
```

und dem Aufruf der beiden gleichen Funktionen innerhalb einer Pfadklammer und dem anschließenden Zeichnen des Pfads auf diese Weise:

```
GpiBeginPath(hps, idPath);
GpiMove(hps, &ptlBeg);
GpiLine(hps, &ptlEnd);
GpiEndPath(hps);
GpiStrokePath(hps, idPath, 0);
```

Es gibt jedoch ganz bedeutende Unterschiede.

Zum einen hat eine mit der normalen GpiLine-Funktion gezeichnete Linie eine sogenannte »kosmetische« Linienbreite. Die Standardbreite der Linie hängt von der Auflösung des Ausgabegeräts ab. Bei einer normalen Linie handelt es sich um eine geräteabhängige Breite. Die Breite der Linie ändert sich nicht, wenn man Matrixtransformationen verwendet, um unterschiedliche Skalierungsfaktoren für den Koordinatenbereich einzustellen. Es gibt zwar im GPI eine Funktion namens GpiSetLineWidth für die Veränderung der kosmetischen Linienbreite, diese ist jedoch in der MS OS/2 Version 1.1 noch nicht implementiert.

Eine Linie, die durch Zeichnen einen Pfads gezogen wird, hat jedoch eine geometrische Breite. Dies ist eine Linienbreite (in Weltkoordinaten), die mit der Funktion GpiSetLineWidthGeom eingestellt wird. Da diese Linienbreite in Weltkoordinaten angegeben wird, wird sie von jeder Skalierung beeinflußt, die man über Matrixtransformationen einstellt.

Zum zweiten kann eine mit GpiLine gezeichnete Linie verschiedene Linientypen aufweisen, die mit der Funktion GpiSetLineType eingestellt werden. Bei diesen Linientypen handelt es sich um verschiedene Kombinationen von Punkten und Strichen. Die Linie wird mit der aktuellen Linienfarbe und der aktuellen Linienmischung gezeichnet.

Eine Linie, die durch Zeichnen eines Pfads gezogen wird, verwendet jedoch den Linientyp nicht. Die Linie wird statt dessen als ein Bereich behandelt, der dem Pfad der Linie folgt, jedoch eine geometrische Breite hat.



◂ *Bild 11: Mit verschiedenen Mustern gefüllte Zeichen*

◂ *Listing 17: VF11.C*

```
/*------------------------------------------------------------
   VF11.C -- Neon font using geometrically-thick lines
  ------------------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Neon (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = " Neon " ;
     static LONG cbText = sizeof szText - 1 ;
     static LONG lForeColor[] = { CLR_DARKRED, CLR_DARKRED, CLR_RED,
                                  CLR_RED,     CLR_WHITE,   CLR_WHITE };
     static LONG lBackColor[] = { CLR_BLACK,   CLR_DARKRED, CLR_DARKRED,
                                  CLR_RED,     CLR_RED,     CLR_WHITE };
     static LONG lWidth[] = { 34, 28, 22, 16, 10, 4 } ;

     INT         iIndex ;
     POINTL      ptl ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;


     ColorClient (hps, cxClient, cyClient, CLR_BLACK) ;

     for (iIndex = 0 ; iIndex < 6 ; iIndex++)
          {
          GpiBeginPath (hps, ID_PATH) ;
          GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Text out
          GpiEndPath (hps) ;

          GpiSetColor (hps, lForeColor[iIndex]) ;
          GpiSetBackColor (hps, lBackColor[iIndex]) ;
          GpiSetBackMix (hps, BM_OVERPAINT) ;

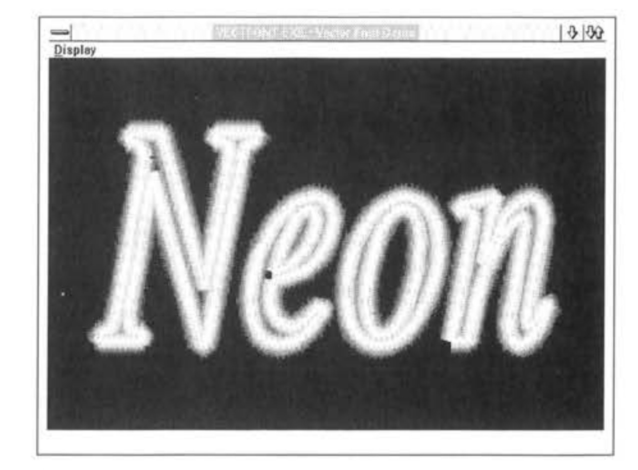
          GpiSetPattern (hps, PATSYM_HALFTONE) ;
          GpiSetLineWidthGeom (hps, lWidth[iIndex]) ;

          GpiStrokePath (hps, ID_PATH, OL) ;            // Stroke path
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;                // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

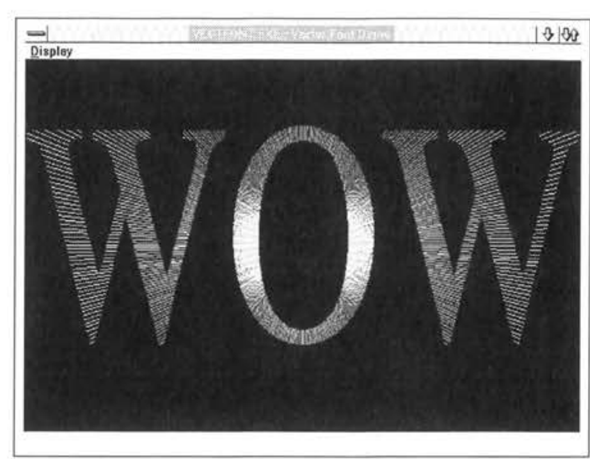
Dieser Bereich wird mit dem Muster gefüllt, das mit GpiSetPattern eingestellt wurde und wird mit den aktuellen Vorder- und Hintergrundfarben und der aktuellen Mustervorder- und -hintergrundmischung gefärbt.

Zum dritten kann eine Linie, die durch Zeichnen eines Pfads gezogen wird, verschiedene Linienverbindungen und -enden haben. Durch Aufruf der Funktion GpiSetLineJoin kann man angeben, daß Linien mit runden, eckigen oder zackigen Verbindungen aufeinandertreffen. Mit GpiSetLineEnd kann man runde, eckige oder flache Linienenden angeben.

Die Funktion in VF12.C (*Listing 19*) demonstriert die Verwendung geometrisch dicker Linien in Verbindung mit Mustern, um den Buchstaben eine Art Neoneffekt zu geben.



▶ *Bild 12: Neon-Zeichen durch Verwendung von dicken Linien*



▶▶ *Bild 13: Farbige Linien in einem Textstring geclippt*

▶ *Listing 18: VF12.C*

```
/*-------------------------------------------------------
   VF12.C -- Fading font with various pattern densities
  -------------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include "vectfont.h"

VOID Display_Fade (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Fade" ;
     static LONG cbText = sizeof szText - 1 ;
     LONG        lPattern ;
     POINTL      ptl ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     GpiSetBackMix (hps, BM_OVERPAINT) ;

     for (lPattern = 8 ; lPattern >= 1 ; lPattern--)
          {
          GpiBeginPath (hps, ID_PATH) ;
          GpiCharStringAt (hps, &ptl, cbText, szText) ;        // Text out
          GpiEndPath (hps) ;

          GpiSetPattern (hps, lPattern) ;
          GpiFillPath (hps, ID_PATH, FPATH_ALTERNATE) ;        // Fill path

          ptl.x += 2 ;
          ptl.y -= 2 ;
          }

     GpiSetPattern (hps, PATSYM_SOLID) ;
     GpiSetBackMix (hps, BM_LEAVEALONE) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;             // Solid

     GpiSetCharSet (hps, LCID_DEFAULT) ;                       // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

▶▶ *Listing 19: VF13.C*

```
/*-----------------------------
   VF13.C -- Clipped Spokes
  -----------------------------*/

#define INCL_GPI
#include <os2.h>
#include <math.h>
#include "vectfont.h"

VOID Display_Spokes (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "WOW" ;
     static LONG cbText = sizeof szText - 1 ;
     static LONG lColors[] = { CLR_BLUE, CLR_GREEN, CLR_CYAN,
                               CLR_RED,  CLR_PINK,  CLR_YELLOW,
                               CLR_WHITE } ;
     double      dMaxRadius ;
     INT         i, iNumColors = sizeof lColors / sizeof lColors[0] ;
     POINTL      ptl ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     ColorClient (hps, cxClient, cyClient, CLR_BLACK) ;

     GpiBeginPath (hps, ID_PATH) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;      // Text string
     GpiEndPath (hps) ;

     GpiSetClipPath (hps, ID_PATH, SCP_AND | SCP_ALTERNATE) ;

     dMaxRadius = sqrt (pow (cxClient / 2.0, 2.0) +
                        pow (cyClient / 2.0, 2.0)) ;
                                                        // Draw spokes
     for (i = 0 ; i < 360 ; i++)
          {
          GpiSetColor (hps, lColors[i % iNumColors]) ;

          ptl.x = cxClient / 2 ;
          ptl.y = cyClient / 2 ;
          GpiMove (hps, &ptl) ;

          ptl.x += (LONG) (dMaxRadius * cos (i * 6.28 / 360)) ;
          ptl.y += (LONG) (dMaxRadius * sin (i * 6.28 / 360)) ;
          GpiLine (hps, &ptl) ;
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;                // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

Sie können diese Funktion durch Auswahl des Menüpunkts »Neon Effect« starten. Die Funktion zeichnet den Pfad unter Verwendung verschiedener geometrischer Linienbreiten gefüllt mit einem PATSYM_HALFTONE-Muster und mehreren Farben. Der Umriß des Fonts ist weiß, er wird jedoch von einem roten Schein umgeben.

Clipping innerhalb von Textzeichen

Die dritte Möglichkeit nach der Anlage eines Pfads besteht in einem Aufruf von GpiSetClipPath:

`GpiSetClipPath(hps, idPath, lOption);`

Der Parameter lOption kann entweder SCP_RESET (was 0L entspricht, also der Standard ist) oder SCP_AND sein. Mit der Option SCP_RESET wird der Clipping-Pfad zurückgesetzt, so daß kein Clipping auftritt. Mit der Option SCP_AND wird der neue Clipping-Pfad auf die Schnittmenge des alten Clipping-Pfads und des gerade in der Pfadklammer definierten Pfads eingestellt. Alle offenen Bereiche im Pfad werden automatisch geschlossen.

Man kann die Option SCP_AND entweder mit SCP_ALTERNATE (Standard) oder SCP_WINDING kombinieren. Wie bei GpiFillPath werden Sie sicherlich den alternativen Modus verwenden, wenn Sie mit Vektorfonts arbeiten.

Die Funktion in VF13.C (*Listing 20*) ruft GpiCharStringAt mit dem Textstring »WOW« innerhalb einer Pfadklammer auf. Darauf folgt ein



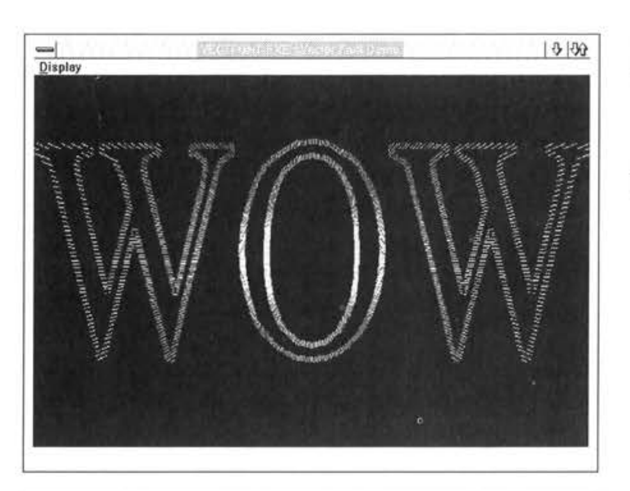


Bild 14: Farbige Wellen im Text geclippt.

Bild 15: Farbige Linien im Outline geclippt

Listing 20: VF14.C

Listing 21: VF15.C



```
/*------------------------------------------------
   VF14.C — Clipped wavy spline curves
  ------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include <stdlib.h>
#include "vectfont.h"

VOID Display_Wavy (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "Hello!" ;
     static LONG cbText = sizeof szText - 1 ;
     static LONG lColors[] = { CLR_BLUE, CLR_GREEN,  CLR_CYAN, CLR_RED,
                               CLR_PINK, CLR_YELLOW, CLR_WHITE } ;
     INT         i ;
     POINTL      ptl, aptl[8] ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn Italic") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     ColorClient (hps, cxClient, cyClient, CLR_BLACK) ;

     GpiBeginPath (hps, ID_PATH) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Text string
     GpiEndPath (hps) ;

     GpiSetClipPath (hps, ID_PATH, SCP_AND | SCP_ALTERNATE) ;

     for (i = 0 ; i < 14 ; i++)
          {
          aptl[0].x = 0 ;
          aptl[0].y = i * cyClient / 14 ;

          aptl[1].x = cxClient / 3 ;
          aptl[1].y = min (cyClient, 2 * i * cyClient / 14) ;

          aptl[2].x = 2 * cxClient / 3 ;
          aptl[2].y = max (0L, (2 * i - 14) * cyClient / 14) ;

          aptl[3].x = cxClient ;
          aptl[3].y = i * cyClient / 14 ;

          aptl[4].x = cxClient ;
          aptl[4].y = (i + 1) * cyClient / 14 ;

          aptl[5].x = 2 * cxClient / 3 ;
          aptl[5].y = max (0L, (2 * (i + 1) - 14) * cyClient / 14) ;

          aptl[6].x = cxClient / 3 ;
          aptl[6].y = min (cyClient, 2 * (i + 1) * cyClient / 14) ;

          aptl[7].x = 0 ;
          aptl[7].y = (i + 1) * cyClient / 14 ;

          GpiSetColor (hps, lColors[i % 7]) ;
          GpiBeginArea (hps, BA_BOUNDARY | BA_ALTERNATE) ;

          GpiMove (hps, aptl) ;                         // Splines
          GpiPolySpline (hps, 3L, aptl + 1) ;
          GpiLine (hps, aptl + 4) ;
          GpiPolySpline (hps, 3L, aptl + 5) ;
          GpiLine (hps, aptl) ;

          GpiEndArea (hps) ;
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;                // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

```
/*------------------------------------------------
   VF15.C — Clipped Spokes
  ------------------------------------------------*/

#define INCL_GPI
#include <os2.h>
#include <math.h>
#include "vectfont.h"

VOID Display_ModSpokes (HPS hps, LONG cxClient, LONG cyClient)
     {
     static CHAR szText[] = "WOW" ;
     static LONG cbText = sizeof szText - 1 ;
     static LONG lColors[] = { CLR_BLUE, CLR_GREEN, CLR_CYAN,
                               CLR_RED,  CLR_PINK,  CLR_YELLOW,
                               CLR_WHITE } ;
     double      dMaxRadius ;
     INT         i, iNumColors = sizeof lColors / sizeof lColors[0] ;
     POINTL      ptl ;

     CreateVectorFont (hps, LCID_MYFONT, "Tms Rmn") ;
     GpiSetCharSet (hps, LCID_MYFONT) ;
     ScaleFontToBox (hps, cbText, szText, cxClient, cyClient) ;
     QueryStartPointInTextBox (hps, cbText, szText, &ptl) ;

     ColorClient (hps, cxClient, cyClient, CLR_BLACK) ;

     GpiBeginPath (hps, ID_PATH) ;
     GpiCharStringAt (hps, &ptl, cbText, szText) ;     // Text string
     GpiEndPath (hps) ;

     GpiSetLineWidthGeom (hps, 6L) ;                    // 1/12 inch
     GpiModifyPath (hps, ID_PATH, MPATH_STROKE) ;
     GpiSetClipPath (hps, ID_PATH, SCP_AND | SCP_ALTERNATE) ;

     dMaxRadius = sqrt (pow (cxClient / 2.0, 2.0) +
                        pow (cyClient / 2.0, 2.0)) ;
                                                        // Draw spokes
     for (i = 0 ; i < 360 ; i++)
          {
          GpiSetColor (hps, lColors[i % iNumColors]) ;

          ptl.x = cxClient / 2 ;
          ptl.y = cyClient / 2 ;
          GpiMove (hps, &ptl) ;

          ptl.x += (LONG) (dMaxRadius * cos (i * 6.28 / 360)) ;
          ptl.y += (LONG) (dMaxRadius * sin (i * 6.28 / 360)) ;
          GpiLine (hps, &ptl) ;
          }
     GpiSetCharSet (hps, LCID_DEFAULT) ;                // Clean up
     GpiDeleteSetId (hps, LCID_MYFONT) ;
     }
```

Aufruf von GpiSetClipPath. Der Clipping-Pfad besteht dann aus dem Inneren der Buchstaben. Die Funktion zeichnet eine Reihe von farbigen Linien, die von der Mitte des Client-Fensters ausgehen. Die Funktion VF14.C (*Listing 20*) verwendet eine ähnliche Technik, zeichnet jedoch eine Reihe von Bereichen, die durch Splines definiert werden.

Veränderung des Pfads

Zwischen dem Aufruf von GpiEndPath zur Beendigung des Pfads und dem Aufruf von GpiStrokePath, GpiFillPath oder GpiSetClipPath kann man noch GpiModifyPath aufrufen. Diese Funktion

verwendet die aktuelle geometrische Linienbreite, -verbindung und das Linienende um jede Linie im Pfad in eine neue Linie umzuwandeln, die einen Bereich um die alte Linie einschließt. Nehmen wir zum Beispiel an, daß der Pfad eine einzige gerade Linie enthält. Nach GpiModifyPath würde der Pfad aus einer geschlossenen Linie in der Form eines Hot Dogs bestehen. Die Breite dieses Hot Dogs ist die geometrische Linienbreite. Die Enden des Hot Dogs können rund, eckig oder flach sein, abhängig vom aktuellen Linienende.

Nach der Anlage eines Pfads sind diese beiden aufeinanderfolgenden Funktionen:

```
GpiModifyPath(hps, ID_PATH, MPATH_STROKE);
GpiFillPath(hps, ID—PATH, FPATH_WINDING);
```

in der Regel identisch mit:

```
GpiStrokePath(hps, ID_PATH, OL);
```

GpiModifyPath und GpiStrokePath sind die beiden einzigen Funktionen, die die geometrische Linienbreite, -verbindungen und -enden verwenden.

Theoretisch kann man GpiStrokePath nach GpiModifyPath aufrufen:

```
GpiModifyPath(hps, ID—PATH, MPATH_STROKE);
GpiStrokePath(hps, ID_PATH, OL);
```

Das sollte eigentlich zu etwas führen, und es müßte auch sehr interessant sein, doch das GPI meldet in der Regel nur, daß es den Pfad nicht erzeugen kann, da er zu komplex ist.

Wir wollen uns statt dessen GpiModifyPath gefolgt von GpiSetClipPath ansehen. Die Funktion in VF15.C (*Listing 22*) ist fast identisch mit der in VF13.C (Listing 20); der Unterschied besteht darin, daß hier die geometrische Linienbreite auf 6 eingestellt (1/2 Zoll) und GpiModifyPath vor GpiSetClipPath aufgerufen wird.

Beachten Sie, daß die farbigen Linien nicht innerhalb des Inneren der Zeichen sondern innerhalb des Umrisses geclippt werden. Durch GpiModifyPath sind die Umrisse der Zeichen selbst in einen Pfad von 1/12 Zoll Dicke umgewandelt worden. Das ist der Pfad, der für das Clipping verwendet wird.

Reicht es?

Ich denke, es ist deutlich geworden, daß die Möglichkeiten, die das GPI für die Arbeit mit Vektorfonts bietet denen von PostScript entsprechen, ja sie teilweise sogar übertreffen. Die GPI-Schnittstelle ist sehr leistungsfähig und vielseitig.

Reicht das? Nein, das ist noch nicht genug. Die Implementierung der Vektorfonts im GPI weist noch einen strukturellen Fehler auf, so daß PostScript weiter an der Spitze bleibt.

Werfen Sie noch einmal einen gründlichen Blick auf Bild 1 und die Anzeige des Helv-Fonts. Sie werden feststellen, daß die beiden Füße des Hs sich in der Breite um einen Punkt unterscheiden, obwohl sie eigentlich gleich breit sein sollten. Dies beruht unzweifelhaft auf einem Rundungsfehler. Es fällt natürlich auf einem Bildschirm mit geringer Auflösung deutlicher auf als auf einem 300-Dpi-Laserdrucker, doch selbst auf einem Laserdrucker beeinträchtigen solche Fehler die Lesbarkeit von Text.

Solche Fehler treten bei PostScript-Fonts nicht auf. PostScript-Fonts sind echte Algorithmen, die Anomalien in der Darstellung von Zeichen erkennen und korrigieren können. Im Gegensatz dazu werden GPI-Fonts (die als einfache Serie von Polylines und Polyfills codiert sind) ohne jedes Feedback und ohne Korrekturmöglichkeit blind gezeichnet.

Wenn wir uns also auch an dem erfreuen können, was das GPI zu bieten hat, gibt es immer noch die Notwendigkeit für Verbesserungen.

Charles Petzold

Keine Angst vor kleinen Tieren:

Die Maus unter OS/2

Keine Angst vor kleinen Tieren, auch wenn das Tierchen für Sie zur unbekannten Art zählt. Von der Maus ist oft im Zusammenhang mit komplexen Bedieneroberflächen wie dem Presentation Manager von OS/2 die Rede, aber das Gerät ist einfach und die Systemaufrufe von OS/2 für die Maus sind nicht schwieriger als die für die Tastatur.

Eine am System angeschlossene Maus steht allen Screen Groups zur Verfügung. In Screen Groups des Presentation Managers betreibt dieser die Maus für alle Programme. In den anderen Screen Groups bleibt es den Programmen überlassen, ob sie die Maus öffnen und benutzen.

Der Ablauf läßt sich in drei Phasen gliedern. Zuerst öffnet ein Programm die Maus und fragt ihre Charakteristika ab; dann bereitet es sie für die Benutzung vor. Schließlich liest es während des Programmlaufs die Mausereignisse der Reihe nach ein und wertet sie aus.

Die Funktionsweise der Maus

Die Maus ist ein kleines Kästchen mit Tasten, das man auf dem Schreibtisch herumschiebt. Die Maushardware erfaßt die Bewegung mechanisch oder optisch und meldet sie dem Computer.

Mausinterrupts und Mausereignisse

Wenn die Maus merkt, daß der Operator sie verschiebt bzw. eine ihrer Tasten drückt oder losläßt, meldet sie das dem Computer mit einem Interrupt.

Die Verarbeitung des Interrupts übernimmt ein Maustreiber, der die Eingaben der Maus einliest und prüft, was passiert ist. Diese Information legt er in einer FIFO-Warteschlange für Mausereignisse ab. Jede Screen Group hat ihre eigene Mauswarteschlange. Der Maustreiber legt die Mausereignisse in die Warteschlange der Screen Group, die momentan im Vordergrund ist.

Er kümmert sich auch um die Position des Mauszeigers auf dem Bildschirm. Selbst wenn der Zeiger unsichtbar ist, aktualisiert der Maustreiber ständig seine Position. Jede Screen Group hat einen eigenen Zeiger; der Zeiger der Screen Group im Vordergrund wird aktualisiert.

Mauseinheiten

Die kleinste Bewegung, die eine Maus erfaßt, heißt Mickey. Wie lang ein Mickey im Verhältnis zu unseren Standardlängeneinheiten ist, variiert je nach Empfindlichkeit der Maus. OS/2 enthält einen Systemaufruf, der die Anzahl der Mickeys pro Zentimeter der aktuellen Maus liefert. Meine Microsoft Busmaus hat 8 Mickeys pro Zentimeter.

Wenn die Maus eine Positionsänderung von mindestens einem Mickey registriert, meldet sie das dem Computer mit einem Interrupt. Je nachdem, ob sie am Systembus hängt oder am seriellen Port, unterbricht sie den Computer direkt oder schreibt ein Byte in den seriellen Port, der dann den Interrupt auslöst.

Da ein Mickey ziemlich klein ist, gibt es viele Interrupts und viele Mausereignisse. Wenn man die Maus aber schnell in einem Zug verschiebt, faßt entweder die Maus oder der Maustreiber

mehrere Mickeys zusammen und meldet nur ein Ereignis.

Wo das geschieht, ist egal. Aufgrund der Vielzahl der Mäuse und Methoden sollte man Maus und Treiber immer als Einheit mit absehbarem Verhalten ansehen.

Der Mauszeiger

Der Maustreiber aktualisiert die Koordinaten der Zeiger aller Screen Groups. In welcher Einheit die Koordinaten festgelegt werden, hängt vom Darstellungsmodus ab. Im Textmodus wird der Zeiger nach Zeilen und Spalten plaziert, im Grafikmodus nach Pixelzeilen und -spalten. Der Zeiger bleibt in allen Modi immer innerhalb des Bildschirms.

Die Zeiger-Zeichenroutine

Bis zu dem Zeitpunkt, wo die Maus eröffnet wird, ist die Zeigerposition nur eine logische Position, man sieht sie auf dem Bildschirm nicht. Erst wenn ein Programm die Maus öffnet, kann es einen sichtbaren Zeiger anfordern. Von da an ruft der Maustreiber die Zeichenroutine nach jeder Positionsänderung des Zeigers auf.

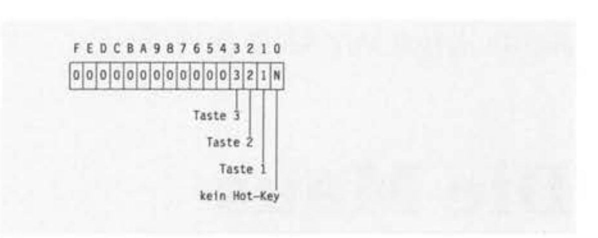
Die Zeichenroutine muß als Unterroutine des Maustreibers laufen. Aus technischen Gründen muß sie selber als Gerätetreiber geladen werden (mit einer device=-Anweisung in der Konfigurations-Datei). Die Standard-Zeigerzeichenroutine heißt POINTDD.SYS und wird als Gerät namens POINTER$ geladen. Man kann auch andere Zeichenroutinen verwenden, aber das Basissystem hat nur eine.

Der Zeiger im Textmodus

Im Textmodus wird der Mauszeiger als Zeichenzelle dargestellt, normalerweise mit invertierten Vorder-/Hintergrundfarben. Wenn der Operator die Maus verschiebt, ruft der Treiber die Zeichenroutine auf, den Zeiger zu löschen und an der neuen Position in den Bildschirmpuffer zu zeichnen.

Dazu wird das Attributbyte einer Zeichenzelle im Bildschirmpuffer verändert. Leider weiß der Scrolling-Code nichts vom Mauszeiger. Wenn der Bildschirm unter dem Zeiger durchscrollt, scrollt die Zeichenzelle mit den invertierten Farben munter mit, und landet in einer anderen Zeile. An der ehemaligen Position des Zeigers steht danach ein anderes Zeichen.

Umgekehrt weiß die Zeigerzeichenroutine nichts vom Scrolling. Wenn sie das nächste Mal aufgefordert wird, den Zeiger zu löschen und neu zu zeichnen, sieht sie nur, daß die Zelle, wo der Zeiger war, sich geändert hat. Sie sucht sich also die neue Zeigerzelle und ändert deren Farben. Beim Scrolling entsteht eine Folge von Zeigerzellen auf dem Bildschirm, deshalb muß man den Mauszeiger vor dem Scrollen ab- und hinterher wieder einschalten.



►► *Bild 1: Die Hot Key-Definition des Session Managers aus MouGetHotKey*

Der Zeiger im Grafikmodus

Die Zeigerzeichenroutine in Version 1.0 arbeitet nur im Textmodus. Im Grafikmodus erscheint einfach kein Zeiger. Mögliche Lösungen siehe unten.

Den Zeiger aussperren

Das Programm kann Rechtecke in Größen zwischen einem Pixel und der Größe des Bildschirms definieren, in denen der Mauszeiger nicht gezeichnet wird. Die Position des Zeigers wird aktualisiert, aber wenn er in dem gesperrten Bereich landet, wird er nicht gezeichnet; er verschwindet am Rand des Bereichs und erscheint auf der anderen Seite wieder. Bewegt man den Zeiger schnell, sieht es aus, als ob er unter dem gesperrten Bereich hindurchgeleitet – der Bereich scheint solide und undurchsichtig.

Wenn man einen rechteckigen Bildschirmausschnitt scrollt und den Zeiger aus dem Bereich ausschließt, wird er nicht mitgescrollt.

Die Maus eröffnen

Die Maus steht jedem Prozeß zur Verfügung, er muß sie nur öffnen. Der Treiber, der die »echte« Maus steuert, erzeugt für jede Screen Group mit MouOpen eine virtuelle Maus – eine Pseudomaus aus Zeigerposition und Warteschlange.

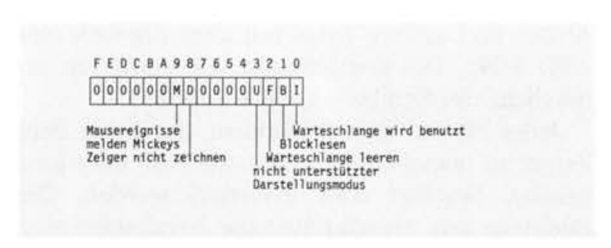
MouOpen eröffnet den Zugriff auf die Maus. Er übernimmt den Namen einer Zeigerzeichenroutine und liefert eine Handle, die die Maus in andere Operationen repräsentiert. Außerdem initialisiert MouOpen die Maus auf einen neutralen Status.

Ist eine Maus angeschlossen?

Möglicherweise ist keine Maus angeschlossen. OS/2 braucht keine Maus und es gibt viele mauslose Systeme. Wenn keine Hardwaremaus existiert oder der Treiber nicht geladen wurde, liefert MouOpen einen Fehlercode.

Die Maus ist Teil der Screen Group, aber manche Prozesse gehören zu keiner – Programme, die mit Abkoppelungs-Befehl gestartet wurden, mit `run=` in der Konfigurations-Datei oder mit DosExecPgm, Option 4 (abgekoppelte Ausführung). Solchen Programmen liefert MouOpen einen Fehlercode.

In der Screen Group des Presentation Managers hat MouOpen sowieso nichts zu sagen und liefert vermutlich auch einen Fehlercode. Der Presentation Manager steuert die Maus für alle Programme, die in seinen Fenstern laufen.



◂◂ *Bild 2: Die Statusbits des Mausgeräts*

Kurz gesagt, ein Programm muß davon ausgehen, daß MouOpen einen Fehlercode ungleich Null liefern kann. Wenn das eintritt, muß das Programm mit dem Operator auf anderem Wege kommunizieren.

Eine Zeichenroutine wählen

MouOpen übernimmt als Eingabeparameter den Namen der Zeigerzeichenroutine. Das ist meist ein far-Zeiger (ein Doppelwort mit Nullen), was zeigt, daß das Programm den Standardsystemzeiger verwendet. Alternativ dazu kann es die Adresse eines Zeichenstrings POINTER$ angeben, die den Namen der Standard-Zeigerzeichenroutine enthält, die als POINTDD.SYS geladen wird.

Version 1.0 kann mit den Grafikmodi nichts anfangen. Für Programme im Textmodus ist das kein Problem, aber für solche im Grafikmodus.

Man könnte (1) eine Ersatzzeichenroutine schreiben, die Pixelgrafik verträgt. Die Routine muß bei der Konfiguration als Gerätetreiber geladen werden, dann kann man MouOpen den Gerätenamen, den sie definiert, angeben. Die Zeigerzeichenroutine arbeitet als direkte Unterroutine des Maustreibers. Der Treiber ruft sie bei einem Interrupt auf. Genau wie andere Treiber muß auch diese Routine im Real und Protected Modus laufen und wird vermutlich in Assembler geschrieben. Das ist nichts für Leute mit schwachen Nerven.

Man kann (2) das Zeichnen des Zeigers auch dem Programm überlassen. Das Programm liest und interpretiert die Mausereignisse sowieso, deshalb ist die Aktualisierung eines kleinen Bildschirmausschnitts bei Mausbewegungen keine große Belastung. Es dauert zwar eine Weile, die Mausbewegung in der Ereigniswarteschlange zu melden, aber der Operator nimmt die Verzögerung kaum wahr.

Die dritte Lösung wäre ein Mausmonitor. Mehr dazu finden Sie im Buch im Kapitel über Monitore.

Der Anfangsstatus

Beim Öffnen wird die Maus initialisiert, der Zeiger auf eine Position in der Bildschirmmitte (bei Zeile 12/Spalte 40 in einer normalen 80x25-Darstellung) gesetzt und der ganze Bildschirm gesperrt, so daß ein Programm den Mauszeiger erst anfordern muß, bevor er sichtbar wird. Der Zeiger hat anfänglich die Standardform (im Textmodus invertierte Farben). Die Ereignismaske, die festlegt, welche Ereignisse in die Warteschlange eingetragen werden, wird so eingestellt, daß alle möglichen Ereignisse eingetragen werden. Dann wird die Warteschlange geleert.

Die Maus vorbereiten

Nachdem die Maus geöffnet wurde, möchte das Programm wissen, was die Maus kann und den Gerätestatus, die Zeigerform und den Proportionalitätsfaktor einstellen.

Alles über die Maus

Verschiedene Systemaufrufe liefern Information über die Maus. MouGetNumButtons meldet die Anzahl der Mausknöpfe (eins, zwei oder drei). Das läßt sich auch in der Maske aus MouGetEventMask erfahren.

Die Empfindlichkeit der Maus erfährt man mit MouGetNumMickeys, der angibt, wieviele Mickeys pro Zentimeter gemessen werden. Das hilft einem beim Einstellen des Proportionalitätsfaktors.

Möglicherweise reserviert der Session Manager bestimmte Tastenkombinationen als Hot Keys. MouGetHotKey liefert die Bitmap aus *Bild 1*. Wenn Bit 0 in dem Wort gesetzt ist, sind keine Tastenkombinationen reserviert und die Anwendung kann alle Maustasten nutzen. Wenn Bit 0 Null ist, zeigt die Kombination der anderen Bits, welche Tastenkombinationen der Session Manager reserviert hat.

Den Gerätestatus abfragen und einstellen

MouGetDevStatus liefert ein Wort mit Statusbits, siehe *Bild 2*. Die Bits 0, 1 und 2 sind für Anwendungen nahezu uninteressant.

Anders Bit 3. Es wird gesetzt, wenn die Zeigerzeichenroutine nicht mit dem aktuellen Darstellungsmodus klarkommt. In Version 1.0 von OS/2 wird das Bit nach MouOpen gesetzt, wenn ein Grafikmodus eingestellt ist.

Wie so oft in OS/2, wird auch der Mausstatus mit einem komplementären Aufrufpaar gepflegt. MouSetDevStatus übernimmt das Wort, das MouGetDevStatus liefert. Man holt sich also den Status, ändert die relevanten Bits und setzt ihn neu.

Nur zwei der Bits aus Bild 2 kann man setzen – Bit 8 verhindert permanent oder vorübergehend, daß der Zeiger gezeichnet wird. Soll der Zeiger wieder gezeichnet werden, muß man das Bit löschen. Mit Bit 9 wählt man zwischen zwei Meldearten. Darüber sprechen wir weiter hinten.

Den Proportionalitätsfaktor abfragen

Der Gerätetreiber wird bei jeder horizontalen oder vertikalen Bewegung der Maus benachrichtigt. Da ein Mickey sehr klein ist, erhebt sich die Frage, ab wievielen Mickeys die Zeigerposition auf dem Bildschirm angepaßt werden soll.

▸ *Bild 3: Die Datenstruktur von MouGet/SetScaleFact*

00h	Mickeys pro Spalte
02h	Mickeys pro Zeile

Die Antwort hängt vom Darstellungsmodus und den Vorstellungen des Operators ab. Der Proportionalitätsfaktor gibt das Verhältnis zwischen Mickeys und der Bewegung des Zeigers auf dem Bildschirm an. MouGetScaleFact liefert den Proportionalitätsfaktor in der Struktur aus *Bild 3*.

Die beiden Werte geben an, um wieviele Mickeys sich die Maus bewegen muß, damit der Mauszeiger eine Einheit verschoben wird. In Textmodi initialisiert MouOpen den Proportionalitätsfaktor der Microsoft Busmaus auf 16 Mickeys pro Zeile und 8 Mickeys pro Spalte. Die Maus muß 16 Mickeys in vertikaler Richtung melden, 2 cm, bevor der Treiber den Zeiger vertikal um eine Zeile verschiebt.

MouSetScaleFact ist die Komplementärroutine; sie übernimmt die Struktur aus Bild 3 und stellt den Proportionalitätsfaktor ein.

Größere Proportionalitätsfaktoren machen die Maus unempfindlicher, kleinere machen sie »lebendiger«. Die Koordinationsfähigkeit zwischen Auge und Hand variiert von Mensch zu Mensch – manche Benutzer können den Mauszeiger anfänglich kaum auf einen bestimmten Punkt des Bildschirms bringen. Später lernen es die meisten. Ich bin selber so ein Zappelphilip und mir ist die Standardeinstellung zu empfindlich. Wenn man die Werte verdoppelt, läßt sich die Sache einfacher handhaben. Der Benutzer sollte die Möglichkeit haben, den Faktor nach seinen Wünschen zu ändern.

▸ *Bild 4: Die Datenstruktur von MouGet/SetPtrShape*

00h	Größe der Bilddaten in Bytes (im Textmodus 4)
02h	Breite des Zeigers in Spalten (im Textmodus 1)
04h	Höhe des Zeigers in Zeilen (im Textmodus 1)
06h	Hot-Spot-Pixelspalte (im Textmodus 0)
08h	Hot-Spot-Pixelzeilen (im Textmodus 0)

Die Zeigerform abfragen und einstellen

Das Bild des Mauszeigers wird durch ein paar Bildbytes plus der Information in der Struktur aus *Bild 4* definiert. Die Bilddaten des Zeigers lassen sich mit MouGetPtrShape abfragen, egal ob der Zeiger gezeichnet wird oder nicht.

MouGetPtrShape liefert die Bildbytes in einen Puffer und füllt die oben gezeigte Struktur aus. Der Komplementäraufruf heißt MouSetPtrShape – er übernimmt dieselben, eventuell aktualisierten, Parameter und installiert sie.

Zum Verständnis der Zeigerdaten muß man eines berücksichtigen: Das Bild des Zeigers ersetzt die Bildschirmdaten nicht. Es wird gezeichnet, indem man die Bildschirmdaten an der Zeigerposition modifiziert. Die Bildbytes spezifizieren die Art der Modifizikation.

Der Mauszeiger wird gezeichnet, indem man Zeigerbildbytes mit Bytes auf dem Bildschirm ANDet und andere Bytes mit dem Ergebnis des AND XORt. Die Kombination der Verfahren ermöglicht vier Ergebnisse, siehe *Tabelle 1.*

Jedes Bit auf dem Bildschirm, das unter dem Zeiger zu liegen kommt, kann auf Null oder Eins gesetzt, bewahrt oder invertiert werden. Die Bildbytes aus MouGetPtrShape beinhalten eine AND-Maske in der Größe des Zeigerbildchens, gefolgt von einer XOR-Maske derselben Größe.

Das Zeigerbild im Textmodus

MouOpen initialisiert das Bild des Textzeigers auf vier Byte:

```
ffh    ffh 00h 77h
```

Die AND-Maske enthält lauter Einser-Bits. Das erste Byte der XOR-Maske enthält lauter 0-Bits, die Bits des Zeichenbytes in der Zelle werden (gemäß Tabelle 1) bewahrt.

▸▸ *Tabelle 1: Die Auswirkungen der AND- und XOR-Masken auf ein Bildbit*

Bit der XOR-Maske		**Null**	**Eins**
Bit der AND-Maske	**Null**	Null	Eins
	Eins	bewahrt	invertiert

Das zweite XOR-Byte invertiert die drei Bits aller Farben. Wenn die Farben Null und Sieben (Weiß auf Schwarz) sind, sind es hinterher Sieben und Null. Das erzeugt nicht etwa inverse Zeichen, sondern einen weißen Block. Bessere Ergebnisse erzielt man durch XORen des zweiten Bytes mit 7fh, das invertiert die Helligkeit und die Farbe des Vordergrunds und so gut wie jedes Zeichen bleibt unter dem Zeiger sichtbar, egal welche Originalfarben es hatte.

Das Zeigerbildchen im Grafikmodus

Im Grafikmodus (vorausgesetzt, der Zeigerzeichencode unterstützt das) wird die Zeigergröße in Pixeln im zweiten und dritten Feld von *Bild 4* angegeben. In dem Fall ist die Form der AND- und XOR-Masken nicht genau definiert. Die Bits in den Masken entsprechen den Pixels des Bildchens, wahrscheinlich Zeile für Zeile geordnet. Es ist nicht bekannt, ob jede Zeile bis zu einer Bytegrenze ausgefüllt werden muß. Es scheint logisch, aber ich konnte es nicht ausprobieren.

Wenn der Grafikmodus mehrere Bitplanes (Bitebenen) erfordert wie die erweiterten Modi von EGA und VGA, gibt es für jede Bitplane ein Maskenpaar in Reihe.

Im Grafikmodus ist das Bild meist größer als ein Pixel (die Einheit, in der die Koordinaten angegeben werden). Es stellt sich also die Frage, an welcher Stelle relativ zu der in den Koordinaten angegebenen Position der Zeiger gezeichnet werden soll. Die letzten beiden Felder aus Bild 4 beantworten die Frage. Sie definieren den sogenannten Hot Spot in relativen Pixelzeilen und Spalten innerhalb des Zeigerbildchens. Dieser Hot Spot wird genau auf dem Pixel, das die Zeigerkoordinaten angeben, gezeichnet, das restliche Bild drum herum.

00h Zeilennummer oben links
02h Spaltennummer oben links
04h Zeilennummer unten rechts
06h Spaltennummer unten rechts

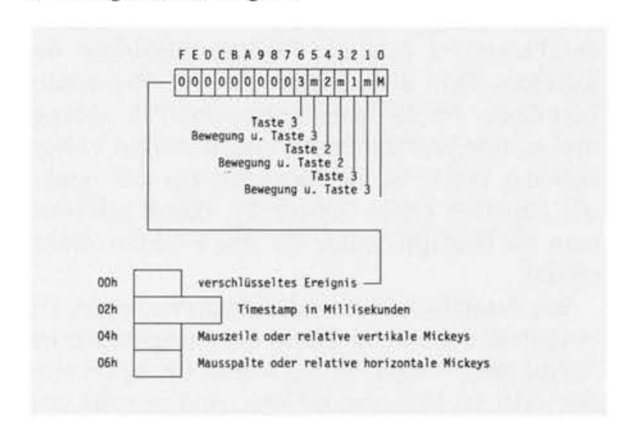
◂◂ *Bild 5: Daten der Sperrbereiche für MouRemovePtr*

Den Zeiger sichtbar machen und aussperren

Der letzte Vorbereitungsschritt für die Arbeit mit der Maus besteht darin, den Zeiger sichtbar zu machen. MouOpen sperrt den Zeiger ja anfänglich aus dem gesamten Bildschirm aus. MouDrawPtr löscht gesperrte Bereiche. Der Zeiger wird – vorausgesetzt, es ist im aktuellen Darstellungsmodus möglich und durch die Gerätestatusbits erlaubt – auf dem Bildschirm gezeichnet und folgt den Mausbewegungen.

Mit MouRemovePtr läßt sich der Zeiger aus Bereichen aussperren. Er wird nicht völlig gelöscht, sondern nur in diesen Bereichen nicht gezeichnet. MouRemovePtr übernimmt die Datenstruktur aus *Bild 5*.

In frühen OS/2-Versionen war die Struktur ungenau dokumentiert. Bild 5 zeigt die Struktur richtig: die Felder 00h und 02h enthalten die Koordinaten der linken oberen Ecke, 04h und 06h die der unteren rechten Ecke. Die Ecken müssen innerhalb der Grenzen des aktuellen Darstellungsmodus liegen.



◂◂ *Bild 6: Die Ereignisdatenstruktur aus MouReadEventQue*

Benutzung der Maus

Sobald die Vorbereitungen abgeschlossen sind, muß das Programm nur noch auf Mausereignisse warten. Wie Sie wissen, legt der Gerätetreiber jedes Mausereignis in einer Warteschlange ab, aus der sich das Programm informiert. Auf den Disketten zu diesem Buch finden Sie ein Programm, das die Maus vorbereitet, auf Ereignisse wartet und die Ereignisse auf dem Bildschirm wiedergibt.

Ereignisse lesen

MouReadEventQue liefert das nächste Ereignis aus der Warteschlange in der Struktur aus *Bild 6*.

Die Bitmask des ersten Wortes gibt Aufschluß über die Art des Ereignisses. Die Bits können in jeder Kombination vorkommen oder alle Null sein. Das weist übrigens nicht auf das Fehlen eines Ereignisses hin, sondern bedeutet, daß eine gedrückte Taste losgelassen wurde.

Ereignisse selektieren

Möglicherweise interessieren eine Anwendung nur bestimmte Ereignisse, beispielsweise nur Bewegungen mit gedrückter Taste. MouGetEventMask liefert ein Wort im Format der Ereignismaske aus *Bild 6* mit einer -1 für jedes mögliche Ereignis. MouOpen setzt in der Maske jedes Ereignis, das die angeschlossene Maus erzeugen kann. Wenn die Maus nur eine oder zwei Tasten hat, werden die Ereignisbits für die fehlenden Tasten auf Null gesetzt.

Man kann die Maske, die MouGetEventMask liefert, ändern und mit MouSetEventMask aktualisieren. Unabhängig von den Ereignissen in der Warteschlange aktualisiert aber der Treiber ständig die Mausposition und zeichnet den Zeiger (falls zugelassen).

Der Ereignis-Timestamp

Die 32-Bit Timestamp im zweiten Feld ist eine Kopie des Millisekundentimers des Systems. Sie gibt den Zeitpunkt an, zu dem das Ereignis in der Warteschlange abgelegt wurde. Der Millisekundenzähler wird bei jedem Timertick oder alle 32 Millisekunden aktualisiert.

Anwendungen verwenden die Timestamp meist zur Bestimmung des Intervalls zwischen zwei Mausereignissen, indem sie die erste Timestamp von der zweiten subtrahieren und das Ergebnis durch 32 teilen (um fünf Bits nach rechts shiften). Genauigkeit geht nicht verloren und man kann das Ergebnis normalerweise in einem nicht vorzeichenbehafteten 16-Bit-Wort speichern. Wenn der Operator zwischen zwei Ereignissen zum Mittagessen geht, überschreitet das Intervall das »Ausdrucksvermögen« eines Wortes. Wenn solche langen Intervalle für das Programm wichtig sind, kann man Intervalle, die das Fassungsvermögen von 16 Bits übersteigen, in 65535 umwandeln und als unendlich bezeichnen.

Die Ereignisposition

Die letzten beiden Felder in Bild 6 enthalten Positionsinformation, gewöhnlich die Zeigerkoordinaten zu dem Zeitpunkt, wo das Ereignis eingetreten ist (also am Ende einer Bewegung).

Wurde Bit 9 des Gerätestatusbytes gesetzt (mit MouSetDevStatus), spiegeln die Ereignisdaten die relativen Mickeys der Bewegung wider. In dem Fall enthalten die Felder vorzeichenbehaftete Integerzahlen, wobei negative Werte »nach oben« oder »nach links« bedeuten, positive »nach rechts« oder »nach unten«. Bei diesem Berichtsmodus werden mehr Ereignisse berichtet. Er eignet sich für Programme, die ihren eigenen Mauszeiger zeichnen.

▸ *Bild 7: Die Datenstruktur aus MouGetQue-Status*

00h	Anzahl der Elemente in der Warteschlange
02h	Maximale Warteschlangenkapazität aus Konfigurations-Datei

Den Warteschlangenstatus prüfen

Mit MouGetNumQueEl findet ein Programm heraus, wie viele Ereignisse in der Warteschlange gespeichert sind. Der Aufruf liefert die beiden Worte aus *Bild 7*. Das erste Wort zeigt, wie viele Ereignisse in der Warteschlange liegen, das zweite gibt an, wie viele hineinpassen. Diese Größe wird in der Konfigurations-Datei eingestellt.

In einem schwach ausgelasteten System hat ein Programm keine Schwierigkeiten, die Maus zu überwachen. Die Auslastung des Systems und die verfügbare CPU-Zeit eines Programms läßt sich aber nicht vorhersehen. Eine Anwendung sollte deshalb die Warteschlange prüfen und falls sie mehr als dreiviertel voll ist, die Priorität des Threads, der die Mausereignisse verarbeitet, erhöhen.

Mit MouFlushQue läßt sich die Warteschlange leeren. MouOpen tut das auch, deshalb ist die Warteschlange am Anfang leer, aber Programme müssen den Vorgang eventuell wiederholen, wenn sie den Arbeitsmodus oder das Bildschirmformat ändern.

Die Zeigerposition ändern

Wenn die Mausereignisse mit ihren absoluten Koordinaten gemeldet werden, weiß das Programm immer, wo der Zeiger steht. Werden sie aber in relativen Mickeys gemeldet, muß es die absolute Zeigerposition mit MouGetPtrPos abfragen. Der Aufruf meldet die aktuellen Zeigerkoordinaten in der Struktur aus *Bild 8*.

MouGetPtrPos meldet unter Umständen eine Position weit von der vorherigen Position des zuletzt gelesenen Ereignisses. Selbst wenn MouGetPtrPos direkt nach MouReadEventQue aufgerufen wird, kann das Programm zwischen zwei Mausereignissen für einen unbekannten Zeitraum eingefroren gewesen sein. In einem schwach ausgelasteten System wird das Intervall kurz sein, aber ein Programm kann nicht voraussetzen, daß es in leicht ausgelasteten Systemen läuft. Wenn die Koordinaten eines Ereignisses wichtig sind, müssen sie in der Ereignisaufzeichnung gemeldet werden.

Mit MouSetPtrPos kann ein Programm neue Zeigerpositionen angeben. Der Aufruf übernimmt die Struktur aus MouGetPtrPos.

Die Maus schließen

Wenn ein Programm die Maus nicht mehr braucht, kann es die Handle mit MouClose für anderweitige Nutzung freigeben. Beim Prozeßende wird MouClose automatisch aufgerufen, wenn die Maus zu dem Zeitpunkt offen ist.

Mausfunktionen ersetzen

Die meisten Mausaufrufe des Systems kann man durch eigenen Code ersetzen. Der Ersatzcode übernimmt die Funktion des ersetzten Codes für die gesamte Screen Group. Ersetzt werden können die Aufrufe aus *Tabelle 2*. Von dieser Möglichkeit machen meist Programme wie der Presentation Manager von OS/2 Gebrauch, die die Ausführungsumgebung anderer Programme verwalten.

Man nennt den Vorgang »einen Ersatz eintragen« (engl. register). Eingetragen wird mit MouRegister, ausgetragen mit MouDeRegister. Man kann mehr als eine Ersatzfunktion auf einmal eintragen, aber es darf immer nur je ein eingetragener Aufruf innerhalb einer Screen Group laufen.

▸▸ *Bild 8: Die Datenstruktur aus MouGet/Set-PtrPos*

00h	Zeilennummer
02h	Spaltennummer

Der Code aller ersetzten Aufrufe liegt in einer einzelnen Prozedur, deren Eintrittspunkt bei jedem Aufruf einer der ersetzten Funktionen irgendwo in der Screen Group aufgerufen wird. Den Stackaufbau dabei zeigt *Bild 9*.

Das Klientenprogramm, das die Mausfunktion aufruft, übergibt ihr Stackparameter. Unterhalb der Parameter liegt die Rücksprungadresse des Klienten, dann zwei Worte, die der Mausrouter (der Code, der die Mausfunktionsaufrufe abfängt und zu den Ersatzaufrufen umleitet) dort ablegt. Eins der Worte ist die Indexnummer der ersetzten Funktion (siehe *Tabelle 2*). Damit selektiert man die Unterprozedur, die eine Funktion wahrnimmt.

Die Ersatzfunktion wird aufgerufen, wenn ein Prozeß in der Screen Group den entsprechenden Aufruf tätigt – egal, ob die Screen Group im Vorder- oder im Hintergrund liegt. Andererseits sind Prozesse in anderen Screen Groups nicht betroffen. Sie verwenden die Standardmausaufrufe bzw. die Ersatzfunktionen, die in ihren jeweiligen Screen Groups eingetragen sind.

Ersatzcode unterliegt einer Reihe von Einschränkungen. Er muß von mehr als einem Programm zugleich betreten werden können – deshalb muß er in einer Dynamic Link Bibliothek liegen, denn nur Dynamic Link (kurz dynalink) Codesegmente können im Adreßraum mehrerer Programme liegen.

Er muß reentrant sein, damit ihn mehr als ein Thread auf einmal ausführen kann (er steht ja mehr als einem Programm zur Verfügung). Wenn er statische Daten verwendet, muß er den Zugriff auf die Daten mit Semaphoren ordnen (nicht mit kritischen Abschnitten, weil die nur innerhalb eines einzelnen Prozesses wirksam sind).

Das geschieht mit MouSynch, der mit DosSemRequest eine Semaphore anfordert, die einer

Screen Group gehört. Der Router löscht die Semaphore, wenn die Ersatzprozedur endet.

Außerdem darf der Code die ersetzten Mausfunktionen selber nicht benutzen und muß deren Operationen durch Aufruf des Maustreibers mit DevIOCTL oder über direkte Hardwarezugriffe aus einem Segment mit I/O-Privileg selber ausführen. Deshalb, aufgrund des variablen Stackaufbaus und der erforderlichen reentrant-Fähigkeit wird die Ersatzfunktion vermutlich in Assembler geschrieben.

David E. Cortesi

◂ Tabelle 2: Die Indexnummern der ersetzten Mausfunktionen

Nummer	Funktion
0	MouGetNumButtons
1	MouGetNumMickeys
2	MouGetDevStatus
3	MouGetNumQueEl
4	MouReadEventQue
5	MouGetScaleFact
6	MouGetEventMask
7	MouSetScaleFact
8	MouSetEventMask
9	MouGetHotKey
10	MouSetHotKey
11	MouOpen
12	MouClose
13	MouGetPtrShape
14	MouSetPtrShape
15	MouDrawPtr
16	MouRemovePtr
17	MouGetPtrPos
18	MouSetPtrPos
19	MouInitReal
20	MouFlushQue
21	MouSetDevStatus

???? Parameterstack für aufgerufene Funktion

Rücksprungadresse zu Klientenprogramm

00?? Indexnummer der ersetzten Funktion

Offset im Code des Mausrouters

Rücksprungadresse zum Mausrouter

◂◂ Bild 9: Der Stackaufbau beim Eintritt in die Ersatz-Mausfunktion

Dieser Artikel ist ein Auszug aus dem »OS/2 Programmierhandbuch« von David E. Cortesi, erschienen im Markt&Technik-Verlag, mit dessen freundlicher Genehmigung der Abdruck erfolgt.

Auch Windows-Anwender brauchen Hilfe:

Hilfe-Verwaltung für Windows

Eine Applikation mit einem integrierten Hilfe-Modul wird schneller erlernt und vielfältiger genutzt und letztendlich aufgrund der besseren Bedienbarkeit auch öfters verkauft. Grund genug, daß jedes Programm ein solches Hilfe-Modul besitzen sollte. Im folgenden wird eine umfassende Hilfeverwaltung vorgestellt, welche mit verhältnismäßig geringem Aufwand in den Quellcode einer Windows-Applikation eingebunden werden kann.

Obwohl Microsoft im Application-Style-Guide des Windows-SDK *[1]* die Implementierung von MDI, Soforthilfe, DDE etc. empfiehlt, wurden solche sinnvolle Features in den Standard-Windows-Applikationen (WRITE, PAINT usw.) bisher nicht integriert. Lediglich der PIF-Editor besitzt eine (etwas halbherzige) Soforthilfe. Das erste Microsoft-Programm, in das eine wirklich gute Soforthilfe integriert wurde, ist MS-Excel.

Ebenso wie bei der MDI-Schnittstelle, die in der letzten Ausgabe des System-Journals ausführlich vorgestellt wurde *[2]*, ist es auch bei der Soforthilfe sinnvoll, sich an der Excel-Implementierung zu orientieren. Obwohl sie an einigen Stellen noch Mängel besitzt, ist sie doch ziemlich konsistent, vielseitig und leistungsfähig. Bevor wir auf die Soforthilfe von Excel eingehen, wollen wir zunächst einmal festhalten, was der Benutzer eigentlich von einer Soforthilfe erwartet und welche Probleme bei der Implementierung auftreten. Anschließend wird dann auf die realisierte Hilfe-Verwaltung eingegangen und ihre Schnittstelle beschrieben. Als Anwendungsbeispiel wird dann eine kleine Applikation mit Namen SHAPE vorgestellt. Im weiteren Verlauf des Artikels wird auf die Implementierung von SHAPE und der applikationsunabhängigen Hilfe-Verwaltung eingegangen. Abschließend erfolgt ein Ausblick auf die Weiterentwicklung der gezeigten Verwaltung.

Allgemeine Forderungen an Applikationen

Der professionelle Anwender von Personal-Computern interessiert sich weniger für technische Einzelheiten der Hardware oder der Software; für ihn ist der PC weitgehend ein nützliches Hilfsmittel zur Lösung seiner Probleme, ähnlich wie viele Autofahrer ihr Fahrzeug als ein reines Fortbewegungsmittel ansehen und sich nicht etwa mit der Einstellung der Ventile beschäftigen wollen. So beschränkt sich die Begeisterung des professionellen Anwenders beim Einsatz von Software meistens auf die Eleganz und Geschwindigkeit, mit der er seine anstehenden Probleme lösen kann. Beispielsweise interessiert es den Anwender *nicht*, daß etwa ein Textverarbeitungsprogramm 50 verschiedene Features für die Gestaltung des Seitenlayouts bietet. Ihn interessiert aber inbesondere, daß er mit dem Programm ein augenblicklich benötigtes Seitenlayout überhaupt erzeugen kann und wie er dies in möglichst kurzer Zeit realisieren kann. Er erwartet, daß ein Programm ihn auf dem Weg zu seiner Problemlösung so weit unterstützt, daß die vom Programm angebotenen Möglichkeiten sein Problem optimal lösen. Dabei muß die Suche nach der Lösung so vom Programm unterstützt werden, daß sie nach kurzer Zeit gefunden wird.

```
/**********************************************************************

 —— S H A P E ———— MS-Windows Application ————————

=======================================================================

This MS-Windows application is a small program for the demonstration
and study of the Help manager. The program permits the drawing of
simple geometric objects in different colors and shapes.

_______________________________________________________________________

Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

Release 1.10 of 89-Jul-13 —— All rights reserved.

**********************************************************************/

/* define/undefine non-debugging option name */
#undef  NDEBUG

#define NOMINMAX
#include <WINDOWS.H>
#include "HELPMGR.H"
#include "DEFS.H"

/* further C standard headers */
#include <STDLIB.H>
#include <STRING.H>
#include <STDIO.H>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)
#define LADDR(r) (LONG)(LPSTR)(r)

/* geometric objects */
enum
 {OBJ_NONE,OBJ_ELLIPSE,OBJ_RHOMBUS,OBJ_SECTOR
 };

/* color index values */
enum
 {COLOR_RED,COLOR_GREEN,COLOR_BLUE
 };

WORD iObject=OBJ_NONE; /* object to paint */

/* size of object in percent parts of client area */
WORD sxObject=50; /* horizontal */
WORD syObject=50; /* vertical */

/* drawing color of object */
DWORD wrgbObject=RGB(0,0,0); /* default is black */

/* brushing for filling geometric object */
LOGBRUSH wlbrObject={BS_HATCHED,RGB(0,0,0),HS_BDIAGONAL};


/**********************************************************************

 ———————————— Application Variables ————————————

**********************************************************************/

BYTE *zAppName="SHAPE"; /* application module name */
BYTE *rzAppTitle;       /* pointer to application title name */
BYTE *rzNoMemory;       /* pointer to error string "no memory" */
HANDLE hiMain;          /* handle to application instance */
HANDLE hAccel;          /* handle to standard accelerator table */


/* ___________
   window handles
   ___________
*/


HWND hwMain; /* handle to main window */


/**********************************************************************
 E r r o r N o M e m
=======================================================================

The error message "Not enough memory" is displayed in a message box.

Parameters:
 hw  is the window handle of the next active window and NULL if no
     window exists.

Used variables:
 rzAppTitle  is the local handle to the application title name.
 rzNoMemory  is the local handle to the no-memory error string.

Return:
 None

**********************************************************************/
```

◂◂ *Listing 1: Das Programm SHAPE.C mit Aufruf der Hilfe-Verwaltung.*

Die im letzten Abschnitt gemachten Äußerungen klingen selbstverständlich. Dennoch genügen viele, auch sehr teuere Programme, diesen Anforderungen nicht. Programme verfügen oft über viele leistungsfähige Features, die aber nicht genutzt werden, weil der Benutzer sie nicht kennt oder sie nicht in adäquater Zeit für sein konkretes Problem anwenden kann. Um solche Programme überhaupt nutzen zu können, sind bei der Neuanschaffung von Software Einführungen und langwierige, teuere Schulungen der Mitarbeiter unvermeidbar. Solche Schulungen übersteigen oft den eigentlichen Anschaffungspreis der Software um ein Vielfaches, ganz abgesehen von dem Arbeitsausfall des Mitarbeiters während der Schulung. Daher eine weitere Behauptung, die jeden Softwareproduzenten herausfordern müßte: Software, die leicht anzuwenden ist und dadurch die Einführung und Schulung verkürzt, kann zu einem höherem Preis verkauft werden, sofern der Anwender Kosten für die Schulung einspart. Außerdem: Je schneller und vielseitiger ein Anwender eine Applikation benutzen lernt, umso schneller und stärker empfiehlt er das Programm seinen Kollegen weiter.

Fensterorientierte Oberflächen wie MS-Windows haben bereits die Benutzerakzeptanz erheblich erhöht. Einheitliche Grundbedienung, Aufklapp-Menüs, Dialogfelder, die WYSIWYG-Darstellung und die Verlagerung »von der Technik zum Problem«, etwa bei der Druckausgabe, erleichtern dem Benutzer den Umgang und die Einarbeitung mit einer neuen Applikation erheblich. Doch bei der Lösung eines konkreten Problems bleibt er ohne weitergehende Features weiterhin alleingelassen. Möchte ein Anwender, der mit der Windows-Applikation MS-Write nicht besonders vertraut ist, in dieser die Randbreite ändern, ohne in die Windows-Dokumentation zu blicken, muß er alle Aufklapp-Menüs durchsuchen, um den dazu erforderlichen Befehl zu finden. Erfordert ein Problem die Ausführung mehrerer Befehle in bestimmter Reihenfolge (beispielsweise: Anlegen eines Inhaltsverzeichnisses in MS-Word), ist dies mit der »Suchmethode« überhaupt nicht lösbar. Soll der häufige Blick in die gedruckte Dokumentation vermieden oder reduziert werden, müssen also der Applikation weitere Features hinzugefügt werden, die zwar nicht zwingend für die Benutzung erforderlich, aber in der Praxis eben doch unverzichtbar sind. Eines dieser Features ist die Soforthilfe, die im Mittelpunkt dieses Artikels steht.

Anforderungen an die Soforthilfe

Die Soforthilfe ist ein Hilfsmittel für den Benutzer zum Lösen seiner individuellen Probleme. Sie enthält Texte, die teilweise ein verkürztes Handbuch darstellen, teilweise aber auch vom Umfang

oder der Präzision der Darstellung dieses erreichen oder gar übertreffen. Aber Information in Form von Texten ist wertlos, wenn die Texte unsystematisch und ungegliedert sind. So gibt es DOS- oder UNIX-Programme, die den Hilfetext als eine große, aber unstrukturierte Einheit zum Lesen beigefügt haben. Sucht man eine bestimmte Lösung, muß man einen Großteil des Textes durchlesen, auch viele augenblicklich unwichtige Stellen. Die Texte in der Soforthilfe müssen also so strukturiert werden, daß der Anwender mit wenig Aufwand »seinen« gewünschten Text findet. Wie muß diese Strukturierung aussehen? Nun, ähnlich wie in einem gedruckten Handbuch. Die Frage des Benutzers lautet: »*Wie* kann ich mein konkretes Problem lösen?«. Er hat für das Problem ein Stichwort (beispielsweise »Randeinstellung« oder »Verzeichnis«), sucht dies im Register, schlägt die passende Seite in der Dokumentation auf und liest sich die entsprechenden Hinweise durch.

Auch bei der Soforthilfe ist ein solche Vorgehensweise nützlich. Es existiert ein Index mit Schlüsselwörtern, der, wenn möglich, alle Fragen umfaßt, die der Benutzer als Problem in (Schlüssel-)Worte fassen kann. Der Benutzer wählt im Index ein Schlüsselwort aus und erhält am Bildschirm sofort den dazu passenden Hilfetext. Die einzelnen Texte sind nicht unzusammenhängend, sondern wie in einem Buch kapitelweise geordnet. Die Soforthilfe sollte den sequentiellen Durchgang durch die Texte anhand dieser kaptitelweisen Anordnung ermöglichen, ähnlich wie ein Benutzer durch ein Buch blättert. Hat der Anwender beispielsweise mit dem Stichwort »Randeinstellung« die Lösung für sein Problem erfahren, so findet er vielleicht das nächste Kapitel »Anordnung von Seitenüberschriften« auch ganz nützlich für sein Problem.

Viele Applikationen realisieren den Index hierarchisch, es gibt beispielsweise Hauptschlüsselwörter und Unterschlüsselwörter. Nachteilig an dieser Methode ist, daß der Benutzer wieder nicht weiß, wo er sein Problem suchen soll. Auch Register in Büchern sind nicht hierarchisch sondern vollständig linear angeordnet. Bei vielen Schlüsselwörtern in einem Hilfe-Index müssen natürlich geeignete Zugriffsmaßnahmen auf den Index am Bildschirm gegeben werden, etwa durch die Eingabe der Anfangsbuchstaben eines Schlüsselworts. Der Vorteil eines hierarchischen Indexes besteht in der schnellen Übersicht, die sich ein Anfänger über das Produkt machen kann. Doch sollte dann dieser Index besser »Inhaltsverzeichnis« genannt werden, und als Alternative zum linearen Index neben diesem existieren.

Zusammenfassend kann also die »Wie«-Soforthilfe als ein Buch betrachtet werden. Sie kann bei entsprechendem Umfang schriftliche Programmbeilagen wie Referenzhandbücher ersetzen. Vorteile ergeben sich aus der flexibleren Hand-

▸▸ *Listing 1: Fortsetzung*

```
VOID ErrorNoMem(HWND hw)

 {HpmSetMsgBoxEnv(0,0); /* no help activation is possible */
  MessageBox(hw,rzNoMemory,rzAppTitle,MB_ICONHAND|MB_OK);
 } /* ErrorNoMem() */

/*********************************************************************
 P r i n t E r r o r
=====================================================================

The error string with resource value <iString> is displayed at screen
in a message box. The environment for the help manager is set.

Parameters:
 hw ...... is the window handle of the next active window and
           NULL if no such window exists.
 iString   is the identification of the error string.
 iDlgItem  is the identification of the message box item which causes
           the error.

Used variables:
 rzAppTitle  is the local handle to the application title name.

Return:
 None

*********************************************************************/

VOID PrintError(HWND hw,WORD iString,WORD iDlgItem)

 {BYTE z[255];

  HpmSetMsgBoxEnv(iDlgItem,iString);
  LoadString(hiMain,iString,z,sizeof(z));
  MessageBox(hw,z,rzAppTitle,MB_ICONASTERISK|MB_OK);
 } /* PrintError() */


/*********************************************************************
 R e a d D l g I t e m W o r d
=====================================================================
This function reads a numerical unsigned integer value in range
0..uMax (including) from the "edit" item <iItem> and stores it at
address <ruDest> if the specification was correct. If so, TRUE is
returned.
In case of any error (bad characters specified or overflow), the error
string STE_NUMVAL is printed in a message box, the input focus is set
to <iItem>, all characters of the field are selected and FALSE is
returned.

Parameters:
 hwBox .. is the window handle of the dialog box.
 iItem .. is the item identifier of the edit field.
 ruDest   is the address of a WORD variable where the
          read value is stored.
 uMax ... is the maximum value which is permitted.

Return:
 If the value is specified correctly, it is stored at address [ruDest]
and
 TRUE is returned. Otherwise FALSE is returned.

*********************************************************************/

BOOL ReadDlgItemWord(HWND hwBox,WORD iItem,WORD *ruDest,WORD uMax)

 {BOOL bTrns;
  WORD u;

  u=GetDlgItemInt(hwBox,iItem,&bTrns,FALSE);
  if (bTrns && u<=uMax)
   {/* no error: store value, return */
    *ruDest=u; return TRUE; /* no error */
   } /* if */
  /* invalid specification or overflow: error message, set focus */
  PrintError(NULL,STE_NUMVAL,iItem);
  /* select all characters in item */
  SendDlgItemMessage(hwBox,iItem,EM_SETSEL,0,MAKELONG(0,32767));
  SetFocus(GetDlgItem(hwBox,iItem));
  return FALSE; /* error */
 } /* ReadDlgItemWord() */


/*********************************************************************
 P a i n t D l g I t e m
=====================================================================

This function paints the contents of the dialog item <iItem>
of dialog box <hwBox>.

Parameters:
 hwBox  is the window handle of the dialog box.
 iItem  is the item identifier of the field to paint.

Used variables:
 rzAppTitle  is the local handle to the application title name.

Return:
 None

*********************************************************************/
```

◂◂ *Listing 1: Fortsetzung*

```
VOID PaintDlgItem(HWND hwBox,HWND iItem)

 {HWND hwItem;
  HDC  hdc;
  RECT wrcItem;
  HBRUSH hbrItem;

  hwItem=GetDlgItem(hwBox,iItem);
  if (!hwItem) return; /* item not available */
  /* repaint contents of item */
  GetClientRect(hwItem,&wrcItem); hdc=GetDC(hwItem);
  InvalidateRect(hwItem,NULL,TRUE);
  /* specify painting brush */
  switch (iItem)
   {case IDTHATCH:
      hbrItem=CreateHatchBrush(HS_BDIAGONAL,wrgbObject);
      break;
    case IDTSOLID:
      hbrItem=CreateSolidBrush(wrgbObject);
      break;
   } /* switch */
  SelectObject(hdc,hbrItem);
  /* draw full contents of window with specified brush */
  Rectangle(hdc,-1,-1,wrcItem.right+1,wrcItem.bottom+1);
  ReleaseDC(hwItem,hdc); DeleteObject(hbrItem);
 } /* PaintDlgItem() */


/**********************************************************************

 ———————————————— Dialog-Box functions ————————————————

**********************************************************************/

/**********************************************************************
 f d S i z e
======================================================================

### Dialog Box function ###

This function processes any messages received
by the DBX_SIZE dialog box.

Parameters:
standard message data (see fwMain)

Return:
standard dialog box function value. DialogBox() returns TRUE.

**********************************************************************/

BOOL FAR PASCAL fdSize(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {WORD sx,sy;

  switch (iMsg)
   {case WM_INITDIALOG:
      SetDlgItemInt(hw,IDHORZ,sxObject,FALSE);
      SetDlgItemInt(hw,IDVERT,syObject,FALSE);
      return TRUE; /* no focus set */
    case WM_COMMAND:
      switch (uP1)
       {case IDOK:
          /* read values in edit fields */
          if (!ReadDlgItemWord(hw,IDHORZ,&sx,100)) break;
          if (!ReadDlgItemWord(hw,IDVERT,&sy,100)) break;
          /* store set values */
          sxObject=sx; syObject=sy;
        case IDCANCEL:
          /* close dialog box */
          EndDialog(hw,uP1);
          return TRUE;
       } /* switch */
      break;
   } /* switch */
  return FALSE;
 } /* fdSize() */


/**********************************************************************
 f d C o l o r
======================================================================

### Dialog Box function ###

This function processes any messages received by
the DBX_COLOR dialog box.

Parameters:
standard message data (see fwMain)

Return:
standard dialog box function value. DialogBox() returns TRUE.

**********************************************************************/

BOOL FAR PASCAL fdColor(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {int i;
  static struct
```

habung eines elektronischen Handbuchs: Schnellerer Zugriff, möglicherweise relational über mehrere Schlüsselwörter, die Aufnahme von benutzereigenen Hilfetexten als eine Art Anmerkung etc.

Ebenso wie ein Referenzhandbuch auf dem Tisch ständig zum Lesen und Suchen verfügbar sein sollte, müssen die Texte der Soforthilfe am Bildschirm zusätzlich zur eigentlichen Applikation anzeigbar sein. Die Texte werden daher sinnvollerweise in einem unabhängigen Fenster angezeigt. Zwar ist bei den heutigen Bildschirmgrößen nicht viel Platz für beide Darstellungen, aber technische Fortschritte auf dem Gebiet der Bildausgabe lassen schon in einiger Zeit großformatige Bildschirme wahrscheinlich machen, die solche Probleme vergessen lassen.

Die Frage nach dem »Was«

Die Frage nach dem »Wie« ist nicht die einzige Art, wie man auf die Soforthilfe zugreifen können möchte. Ein dialogorientiertes Betriebssystem zeigt dem Anwender eine Menge Dialogobjekte, deren Bedeutung er als Anfänger zunächst nicht kennt. Viele davon erlauben ihm aber Probleme zu lösen, die er vielleicht als solche im Rahmen seiner Computer-Anwendung noch gar nicht gesehen hat. Andererseits besitzen oft »Experten« zu vielen Dialogobjekten ein gewisses »Halbwissen«: Man kennt zwar die Bedeutung des Objekts, die Einzelheiten aber hat man schon wieder vergessen. Es muß daher einen zweiten Zugriff auf die Hilfetexte geben, nämlich über die Frage »*Was* bedeutet dies?«. Der Anwender muß das gewünschte Element mit der Maus oder Tastatur anklicken und die Hilfeverwaltung zeigt ihm einen passenden Hilfetext mit der Bedeutung des Elements an, gegebenenfalls mit Anwendungsbeispielen angereichert. Diese Art der Soforthilfe wird häufig als »kontextorientierte Hilfe« bezeichnet.

Eine automatische aber verkürzte Anzeige der Bedeutung von Befehlen schreibt die IBM-SAA-Spezifikation vor: Jede Applikation besitzt am unteren Fensterrand eine Statuszeile, die beim Auswählen von Befehlen innerhalb eines Aufklapp-Menüs eine kurze Beschreibung des Befehls anzeigt, die zwar ausführlicher als der Befehlsname im Menü ist, aber natürlich nicht umfangreichere Hilfetexte ersetzen kann. Dennoch ist diese Einzeilen-Darstellung ein nicht zu unterschätzendes Hilfsmittel.

Die Frage nach dem »Warum«

Ein weiteres Kapitel von Ärgernissen für Benutzer stellen Fehlermeldungen dar. Angefangen von Meldungen für Genies wie »Fehler! Programm beendet« über Meldungen für Gedächnis-

▸▸ *Listing 1: Fortsetzung*

künstler wie »Fehler XUA4315« zu so aussagekräftigen Hinweisen wie »Ungültiger Pfad, kein Verzeichnis oder Verzeichnis nicht leer« ist alles vertreten. Obwohl ein Programm meistens sehr genau »weiß«, warum etwas falsch ist, hat der Implementierer aus Bequemlichkeit eine detaillierte Ausgabe vernachlässigt. So kann es etwa sein, daß man im Handbuch bei der Beschreibung des Fehlers »XUA4315« mehrere Hinweise erhält, wie der Fehler verursacht sein könnte, die beim unerfahrenen Benutzer Erinnerungen an das Orakel von Delphi auslösen. Obwohl das Programm eine detaillierte Fehlerausgabe aufgrund seiner Informationen bieten könnte, bleibt sie dem Anwender verborgen. Die Soforthilfe muß also auch in diesem Fall Unterstüzungen für den Benutzer anbieten.

Bei Windows-Applikationen werden Fehlermeldungen in Hinweisfeldern (im Original *message boxes*) angezeigt. Oftmals ist der kurze Hinweis für den erfahrenen Benutzer ausreichend, etwa »Format falsch« oder »Datei nicht gefunden«. Es gibt wenig Sinn, jedesmal eine langatmige Erklärung aufzuführen, warum etwa das Format einer eingegebenen Zahl fehlerhaft ist. Doch der Anfänger ist häufig mit den kurzen Fehlermeldungen überfordert. Seine Frage »*Warum* dieser Fehler?« sollte von der Hilfeverwaltung mit ausführlicheren Sätzen erläutert werden, wobei im Idealfall auf die tatsächlichen Ursachen des Fehlers eingegangen wird. Eine detaillierte und vollständige Unterstützung des Anwenders in solchen Fragen erleichtert die Einarbeitung in ein neues Programm erheblich. Auch das die Software entwickelnde Unternehmen wird entlastet, wenn typische Fragen und Antworten an die »Hotline« von Anfang an in die Soforthilfe übertragen werden.

Heutige Realisierungen von Soforthilfen

Nach dem etwas ausführlichen Ausflug in die Ergonomie der Soforthilfe soll kurz untersucht werden, inwieweit die Soforthilfe bereits bei heute verfügbaren Applikationen integriert ist. Unabhängig von MS-Windows lassen sich bei der Soforthilfe die Applikationen weitgehend in folgende vier Klassen einteilen:

- Programme ohne jede Soforthilfe. Hierzu zählen beispielsweise DOS-Command, MS-Write und alle DOS-Systemprogramme.
- Programme mit ein »bißchen« Soforthilfe. Hierbei werden einige wenige Hinweise auf die Möglichkeiten der Applikation gegeben, beispielsweise welche Befehle vorhanden sind oder wie Parameter von Befehlen angegeben werden müssen. Zu solchen Applikationen zählen beispielsweise PC-Tools (Version 4) oder Norton-Utilities.
- Programme mit ausführlichen Hilfetexten, die

```
 {int  vColor;
  HWND hw;
 }
Entry[3];
BYTE z[5];
DWORD uL;
int vColor;

switch (iMsg)
 {case WM_INITDIALOG:
    uL=wrgbObject; /* current set color */
    for (i=COLOR_RED;i<=COLOR_BLUE;i++)
     {/* set scroll range, scroll value and percent values */
      Entry[i].vColor=(100*(BYTE)uL+128)/255;
      uL>>=8; /* move to next color */
      Entry[i].hw=GetDlgItem(hw,IDRED+i);
      sprintf(z,"%d%%",Entry[i].vColor);
      SetDlgItemText(hw,IDVRED+i,z);
      SetScrollRange(Entry[i].hw,SB_CTL,0,100,FALSE);
      SetScrollPos(Entry[i].hw,SB_CTL,Entry[i].vColor,TRUE);
     } /* for */
    return TRUE; /* no focus set */
  case WM_CTLCOLOR:
    switch (GetWindowWord(P2LO,GWW_ID))
     {case IDTRED:
        SetTextColor(uP1,RGB(255,0,0)); break;
      case IDTGREEN:
        SetTextColor(uP1,RGB(0,255,0)); break;
      case IDTBLUE:
        SetTextColor(uP1,RGB(0,0,255)); break;
      default:
        return NULL;
     } /* switch */
    return (BOOL)GetStockObject(NULL_BRUSH);
  case WM_HSCROLL:
    for (i=COLOR_RED;i<=COLOR_BLUE;i++)
     {if (P2HI!=Entry[i].hw) continue;
      vColor=Entry[i].vColor;
      switch (uP1)
       {case SB_TOP:
          vColor=0; break;
        case SB_BOTTOM:
          vColor=100; break;
        case SB_LINEUP:
          if (vColor!=0) vColor—;
          break;
        case SB_LINEDOWN:
          if (vColor<100) vColor++;
          break;
        case SB_PAGEUP:
          if (vColor>10) vColor-=10; else vColor=0;
          break;
        case SB_PAGEDOWN:
          if (vColor<90) vColor+=10; else vColor=100;
          break;
        case SB_THUMBPOSITION:
        case SB_THUMBTRACK:
          vColor=P2LO; break;
       } /* switch */
      if (vColor!=Entry[i].vColor)
       {/* set new values */
        Entry[i].vColor=vColor;
        SetScrollPos(Entry[i].hw,SB_CTL,vColor,TRUE);
        sprintf(z,"%d%%",Entry[i].vColor);
        SetDlgItemText(hw,IDVRED+i,z);
       } /* if */
     } /* for */
    break;
  case WM_COMMAND:
    switch (uP1)
     {case IDOK:
        /* store set color values */
        wrgbObject=RGB
         ((Entry[COLOR_RED].vColor*255+50)/100,
          (Entry[COLOR_GREEN].vColor*255+50)/100,
          (Entry[COLOR_BLUE].vColor*255+50)/100
         );
        wlbrObject.lbColor=wrgbObject;
      case IDCANCEL:
        /* close dialog box */
        EndDialog(hw,uP1); return TRUE;
     } /* switch */
    break;
 } /* switch */
 return FALSE;
} /* fdColor() */

/***********************************************************************
 f d B r u s h
========================================================================
### Dialog Box function ###

This function processes any messages received by
the DBX_BRUSH dialog box.

Parameters:
standard message data (see fwMain)

Return:
standard dialog box function value. DialogBox() returns TRUE.

***********************************************************************/
```

◂◂ *Listing 1: Fortsetzung*

```
BOOL FAR PASCAL fdBrush(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {static WORD iStyle; /* current selected style */

  switch (iMsg)
   {case WM_INITDIALOG:
      switch (wlbrObject.lbStyle)
       {case BS_HATCHED:
          iStyle=IDHATCH; break;
        case BS_SOLID:
          iStyle=IDSOLID; break;
        default:
          iStyle=IDHOLLOW; break;
       } /* switch */
      CheckRadioButton(hw,IDHOLLOW,IDSOLID,iStyle);
      return TRUE; /* no focus set */
    case WM_PAINT:
      PaintDlgItem(hw,IDTHATCH); PaintDlgItem(hw,IDTSOLID);
      break;
    case WM_COMMAND:
      switch (uP1)
       {case IDHOLLOW:
        case IDHATCH:
        case IDSOLID:
          if (P2HI==BN_CLICKED)
           {iStyle=uP1; CheckRadioButton(hw,IDHOLLOW,IDSOLID,iStyle);
           } /* if */
          break;
        case IDOK:
          /* store set value */
          switch (iStyle)
           {case IDHOLLOW:
              wlbrObject.lbStyle=BS_HOLLOW; break;
            case IDHATCH:
              wlbrObject.lbStyle=BS_HATCHED; break;
            case IDSOLID:
              wlbrObject.lbStyle=BS_SOLID; break;
           } /* switch */
        case IDCANCEL:
          /* close dialog box */
          EndDialog(hw,uP1); return TRUE;
       } /* switch */
      break;
   } /* switch */
  return FALSE;
 } /* fdBrush() */


/**********************************************************************
 R e a d D i a l o g
======================================================================

This function creates a modeless dialog box, displayes it at screen
and returns if the user closes the box.
The function returns TRUE if the function DialogBox() returns IDOK and
FALSE otherwise. If the dialog box is not created because the memory
is too small, a message box with an error is created and FALSE is
returned.

Parameters:
 fd .. is the dialog box function.
 uBox  is the number of the dialog box (DBX_...-key).

Return:
 TRUE if the dialog box was created and the OK-button was pressed to
 close it or FALSE if the CANCEL-button was pressed or no box was
 created.

**********************************************************************/

BOOL ReadDialog(WORD uBox,FARPROC rfd)

 {FARPROC rfdInst;
  int v;

  rfdInst=MakeProcInstance(rfd,hiMain);
  HpmSetDlgBoxEnv(uBox); /* set identification of current box */
  if (rfdInst!=NULL)
   {v=DialogBox(hiMain,MAKEINTRESOURCE(uBox),hwMain,rfdInst);
    FreeProcInstance(rfdInst);
    if (v!=-1)
     {/* box was created: return TRUE if OK-button pressed */
      return v==IDOK;
     } /* if */
   } /* if */
  /* procedure-instance or box not created: print error message */
  ErrorNoMem(hwMain); return FALSE;
 } /* ReadDialog() */


/**********************************************************************
 C m d M a i n
======================================================================
The command code <iCmd> of the main window is executed. Here only
command codes from the system menu (CMD_...) are executed. Commands
from child windows or similiar sources are not analysed here.

Parameters:
 iCmd  is the command code.
Return:
 TRUE if the command was consumed, FALSE if not.
**********************************************************************/
```

über Schlüsselwörter aufrufbar sind, aber mit insgesamt unzureichendem Zugriff. Es fehlt eine ausgereifte kontextorientierte Soforthilfe. Der Index mit den Schlüsselwörtern ist oft ziemlich klein und darüber hinaus merkwürdig hierarisch organisiert, so daß man sich über mehrere Ebenen an sein Problem herantasten muß. MS-Word oder PageMaker von Aldus sind Beispiele für solche Programme. Bei beiden kann der Hilfetext nicht parallel angezeigt werden, während man Befehle ausführt. So muß man einigen Hinweisen den Hilfetext (etwa beim Erstellen von Druckformatvorlagen) erst von Hand abschreiben oder ausdrukken, ehe man die empfohlene Anweisungsreihenfolge ausführen möchte, außer man rechnet sich zu *den* Anwendern, die problemlos die Ausführungsreihenfolge zehn komplexer Anweisungen im Kopf behalten können.

• Programme mit umfangreicher Soforthilfe, die sowohl über den Index- oder Verzeichniszugriff zu den Hilfetexten verfügen als auch über die Kontexthilfe. Sie kann sowohl vom Anfänger zum Erlernen bestimmter Fähigkeiten einer Applikation als auch vom Experten zum Nachschlagen selten benutzer Features der Applikation eingesetzt werden. Zu solchen Programmen zählen *DOS-4-Shell*, MS-Excel oder Quick-C (Version 2.0).

Abgesehen davon wurden etwa bei MS-Word oder Excel weitere Hilfefunktionen eingefügt, die unter Begriffen wie »Lernprogramm« oder »Leitfaden« dem Anwender dienen sollen. Solche Ergänzungen beruhen meist auf externen Programmen, die unter dem betreffenden Programm aufgerufen werden. Um diesen Beitrag nicht zu lang werden zu lassen, wird auf die Realisierung dieser an sich sehr nützlichen Ergänzungen nicht weiter eingegangen. Im folgenden wollen wir uns detaillierter mit der Excel-Soforthilfe beschäftigen, die im weiteren als Vorbild genommen werden soll.

Die Fähigkeiten der Excel-Soforthilfe

Die Hilfe-Funktion von MS-Excel orientiert sich am Application-Style-Guide des Windows-SDK *[1]*. Die Taste F1 ist für den Hilfeaufruf reserviert, er kann aber auch über das Hilfemenü am rechten Ende der Menüleiste erfolgen. Es gibt zwei Arten von Hilfen: Die indexorientierte Hilfe über den »Index«-Befehl im Menü und die kontextorientierte Hilfe über die Tastenkombination Shift-F1 innerhalb eines Menüaufrufs oder Dialogfelds. Erstere beantwortet das »Wie«, die zweite das »Was« und »Warum«. Leider ist der Index hierarchisch aufgeteilt, so daß man Mühe hat, den Text zu einem bestimmten Problem zu finden. Es werden auch nicht alle Features von Excel in den Hilfetexten erläutert, so fehlen etwa Informationen über die Makrofunktionen. Man

▸▸ *Listing 1: Fortsetzung*

kann also nicht ohne zusätzliches Excel-Referenzhandbuch arbeiten.

Als weiteres ermöglicht die kontextorientierte Hilfe die Abfrage der Bedeutung von Menübefehlen oder anderer Dialogelemente, beispielsweise am Fensterrand. Hierzu wird nach der Eingabe von Shift-F1 der Mauszeiger mit einem zusätzlichen Fragezeichen versehen. Sobald ein Menü oder ein anderes Dialogelement angeklickt wird, wird nicht die übliche Aktion ausgeführt, sondern statt dessen ein Hilfetext angezeigt, der die Aktion erklärt. Beim Aufruf von Befehlen im Menü erfolgt bei Excel automatisch eine Kurzbeschreibung des jeweils markierten Befehls in der Statuszeile, wie dies die SAA-Richtlinie vorschreibt.

Die Hilfetaste F1 kann auch aktiviert werden, wenn ein Menüpunkt ausgewählt oder ein Dialog- beziehungsweise Hinweisfeld angezeigt wird. In diesem Fall erläutert die Hilfeverwaltung den markierten Menübefehl oder das angezeigte Dialogfeld. Bei Hinweisfeldern (etwa nach Eingabefehlern) wird der Fehler detaillierter erläutert (Frage nach dem »warum«). Leider enthält Excel keine Hilfefunktion für die einzelnen Einträge innerhalb eines Dialogfelds. Man muß vielmehr langatmig den gesamten Text für ein Dialogfeld durchlesen, um Hinweise auf die einzelnen Dialogfelder zu erhalten. Bei größeren Dialogfeldern mit komplizierteren Auswahlmöglichkeiten ist dies ziemlich »nervig«. Dies ist ein Schönheitsfehler, der noch beseitigt werden sollte.

Andererseits entschädigt die Excel-Hilfeverwaltung mit interessanten Features bei der Anzeige des Hilfetextes. Diese erfolgt in einem eigenständigen Fenster, das ständig neben der Applikation gehalten werden kann und sich sogar zu einem Sinnbild verkleinern läßt. Die Größe des Fensters kann vom Anwender auf seine Bildschirmgröße angepaßt werden, das Format der Hilfetexte paßt sich entsprechend an (dynamischer Zeilenumbruch). Glossarbegriffe sind punktiert unterstrichen, mit *einem* Tastendruck lassen sich die Definitionen solcher Begriffe erhalten. Unterstrichene Querverweise, erlauben den schnellen Wechsel zwischen den einzelnen Hilfetexten, indem jene einfach mit der Maus angeklickt werden.

Die Excel-Hilfe stellt sicherlich eines der am weitesten fortgeschrittenen Hilfeverwaltungs-Konzepte dar. Es ist unmöglich, in einer Implementierung für einen Zeitschriftenartikel alle Eigenschaften zu übernehmen. Es wird daher vor allem Wert auf die Schnittstellen zu der Applikation und zum Windows-System gelegt, weitergehende Features wie Glossarbegriffe, Querverweise, Fenster mit Textumbruch wurden dagegen nicht realisiert; sie stellen »Fleißarbeiten« innerhalb der eigentlichen Hilfeverwaltung dar, die den Rahmen dieses Artikels sprengen würden.

Im folgenden Abschnitt wollen wir allmählich

```
BOOL CmdMain(int iCmd)

 {switch(iCmd)
   {case CMD_ELLIPSE:
      iObject=OBJ_ELLIPSE; break;
    case CMD_SECTOR:
      iObject=OBJ_SECTOR; break;
    case CMD_RHOMBUS:
      iObject=OBJ_RHOMBUS; break;
    case CMD_SIZE:
      if (!ReadDialog(DBX_SIZE,fdSize)) return TRUE;
      break;
    case CMD_COLOR:
      if (!ReadDialog(DBX_COLOR,fdColor)) return TRUE;
      break;
    case CMD_BRUSH:
      if (!ReadDialog(DBX_BRUSH,fdBrush)) return TRUE;
      break;
    case CMD_CLEAR:
      iObject=OBJ_NONE; break;
    default:
      /* other commands: not consumed */
      return FALSE;
  } /* switch */
  /* redraw object */
  InvalidateRect(hwMain,NULL,TRUE); return TRUE;
 } /* CmdMain() */


/*******************************************************************

 ---------------------- Main Window function ----------------------

*******************************************************************/
LONG FAR PASCAL fwMain(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {PAINTSTRUCT wps;
  HDC hdc;
  HPEN hpnOld;
  HBRUSH hbrOld,hbrFrame;
  POINT pCenter;
  int sxHalf,syHalf;
  RECT wrc;
  LONG ulRet;
  HRGN hrgnE,hrgnS;
  HCURSOR hcrSave;

  switch (iMsg)
   {case WM_CREATE:
      /* set application window handle */
      hwMain=hw; return OL;
    case WM_SIZE:
      if (uP1!=SIZEICONIC) HpmSetCmdLine(P2HI);
      break;
    case WM_PAINT:
      hdc=BeginPaint(hw,&wps);
      hpnOld=SelectObject(hdc,CreatePen(0,1,wrgbObject));
      hbrOld=SelectObject(hdc,CreateBrushIndirect(&wlbrObject));
      GetClientRect(hw,&wrc);
      pCenter.x=wrc.right/2; pCenter.y=wrc.bottom/2;
      sxHalf=(pCenter.x*sxObject)/100;
      syHalf=(pCenter.y*syObject)/100;
      switch (iObject)
       {case OBJ_ELLIPSE:
          Ellipse
           (hdc,pCenter.x-sxHalf,pCenter.y-syHalf,
            pCenter.x+sxHalf,pCenter.y+syHalf
           );
          break;
        case OBJ_RHOMBUS:
         {POINT mpRhombus[4];
          mpRhombus[0].x=pCenter.x-sxHalf; mpRhombus[0].y=pCenter.y;
          mpRhombus[1].x=pCenter.x; mpRhombus[1].y=pCenter.y-syHalf;
          mpRhombus[2].x=pCenter.x+sxHalf; mpRhombus[2].y=pCenter.y;
          mpRhombus[3].x=pCenter.x; mpRhombus[3].y=pCenter.y+syHalf;
          Polygon(hdc,mpRhombus,4); break;
         } /* case */
        case OBJ_SECTOR:
          /* longer delay: set hour glass cursor */
          hcrSave=SetCursor(LoadCursor(NULL,IDC_WAIT));
          /* create region of elliptic part (very slow!) */
          hrgnE=CreateEllipticRgn
           (pCenter.x-sxHalf,pCenter.y-syHalf,
            pCenter.x+sxHalf,pCenter.y+3*syHalf
           );
          /* create region of rectangle part */
          hrgnS=CreateRectRgn
           (pCenter.x-sxHalf,pCenter.y-syHalf,
            pCenter.x+sxHalf,pCenter.y+syHalf
           );
          /* combine both regions and draw cotents of regions */
          CombineRgn(hrgnS,hrgnE,hrgnS,RGN_AND); PaintRgn(hdc,hrgnS);
          /* draw frame */
          hbrFrame=CreateSolidBrush(wrgbObject);
          FrameRgn(hdc,hrgnS,hbrFrame,1,1); DeleteObject(hbrFrame);
          SetCursor(hcrSave); /* set old cursor */
          break;
      } /* switch */
     /* select default painting objects, delete created objects */
     DeleteObject(SelectObject(hdc,hpnOld));
     DeleteObject(SelectObject(hdc,hbrOld));
     EndPaint(hw,&wps); return OL;
```

```
    case WM_COMMAND:
      if (uP1!=CMD_EXIT)
       {/* user control command */
        if (CmdMain(uP1))
         {/* command executed, but help manager needs it too */
          return HpmProcessHelpMsg(hw,iMsg,uP1,ulP2);
         } /* if */
        break; /* not consumed */
       } /* if */
      /* continue */
    case WM_CLOSE:
      DestroyWindow(hw); return OL;
    case WM_DESTROY:
      PostQuitMessage(0); return OL;
   } /* switch */
  ulRet=HpmProcessHelpMsg(hw,iMsg,uP1,ulP2);
  if (ulRet!=OL) return ulRet;
  return DefWindowProc(hw,iMsg,uP1,ulP2);
 } /* fwMain() */


/**********************************************************************

 ——— Window dependent initialization (one for all instances) ———

**********************************************************************/

/**********************************************************************
 C r e a t e C l a s s
======================================================================

The classes of all application specific windows are registered.

Parameters:
 none

Return:
 TRUE   if all classes created.
 FALSE  if any error during creation (memory error).

**********************************************************************/

BOOL CreateClass(VOID)

 {WNDCLASS wwc;

  /* —— Create window class of Application —— */
  wwc.lpszClassName=zAppName; wwc.hInstance=hiMain;
  wwc.lpfnWndProc=fwMain;
  wwc.hCursor=LoadCursor(NULL,IDC_ARROW);
  wwc.hbrBackground=GetStockObject(WHITE_BRUSH);
  wwc.style=CS_HREDRAW|CS_VREDRAW;
  wwc.hIcon=NULL;
  wwc.lpszMenuName=MAKEINTRESOURCE(MNU_MAIN);
  wwc.cbClsExtra=0; wwc.cbWndExtra=0;
  /* register window, return if error */
  if (!RegisterClass(&wwc)) return FALSE;
 } /* CreateClass() */


/**********************************************************************

 ——— Instance dependent initialization (for every instance) ———

**********************************************************************/
/**********************************************************************
 C r e a t e A p p W i n d o w
======================================================================

All globally existing windows are created by this function.

Parameters:
 none

Return:
TRUE is returned if all data read, otherwise FALSE
(memory too small).

**********************************************************************/
BOOL CreateAppWindow(VOID)

 {/* application's window, return if error */
  if (!CreateWindow
   (zAppName,rzAppTitle,WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN,
    CW_USEDEFAULT,0,CW_USEDEFAULT,0,NULL,NULL,hiMain,NULL)
   )
    return FALSE;
  return TRUE;
 } /* CreateAppWindow() */


/**********************************************************************
 I n i t I n s t a n c e
======================================================================
### The main init module entry point ###
The complete initialization of a new instance of the application
window. If no previous instance exists, the application window is
initialized and the common data for all instances are created.
Otherwise, common data are copied from the previous instance to the
new instance. All individual data in the instance data segment are
initialized. If memory is too small, a error message box is displayed
and FALSE is returned.
```

◂◂ *Listing 1: Fortsetzung*

zur Implementierung der Hilfeverwaltung gelangen, indem wir die Anwenderseite verlassen und mehr die Probleme untersuchen, die den Programmierer im Zusammenhang mit der Soforthilfe betreffen.

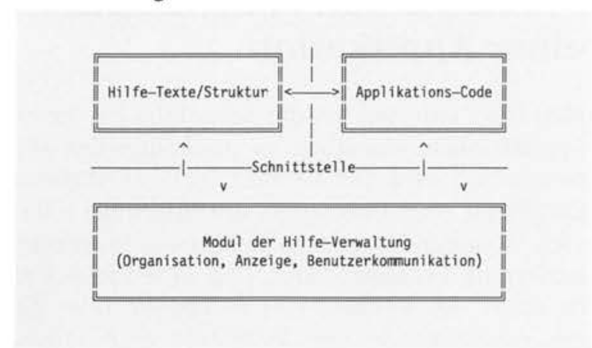
Probleme beim Entwickeln einer Applikation

Man fragt sich, warum die Soforthilfe bei vielen Applikationen eigentlich so stiefmütterlich implementiert wird. Der Grund liegt darin, daß der Entwickler eines Programms die Soforthilfe selbst nicht benötigt, da er natürlich »sein« Programm auswendig bedienen kann. Für viele Entwickler ist daher der Einbau einer Soforthilfe oder die Zusammenstellung der Hilfetexte eine lästige Aufgabe, die sie weitgehend ausklammern möchten. Erst bei entsprechendem Marktdruck, wenn die Forderungen der Kundschaft nach einer Hilfefunktion lauter werden, wird dann oft spontan und ohne große Überlegungen eine mehr oder weniger schlechte Soforthilfe integriert. Die Hilfetexte sind unzusammenhängend oder unvollständig, der Index enthält nicht die Hinweise, die der Anwender letztendlich sucht. Gute Software-Entwickler sind nicht zwingend auch gute Lehrmeister für ihr Produkt, weil sie dieses zu sehr aus ihrer Implementierersicht betrachten und sich nicht ausreichend in den zunächst unbedarften und naiven Benutzer hineinversetzen können.

Größere Software-Häuser haben diesen Widerspruch frühzeitig erkannt und beispielsweise das Schreiben der Handbücher von der Entwicklung der Software personell getrennt. Entsprechend sollte auch die Strukturierung der Soforthilfe und die Gestaltung ihrer Texte auf ergonomisch sensible Mitarbeiter übertragen werden, die weder programmieren können müssen, noch mit der internen Realisierung eines Produkts vertraut sein müssen (oder sollten). Für diese Trennung der Programmierung einer Applikation einerseits und der Gestaltung der Soforthilfe andererseits ist eine klare Schnittstelle vorzusehen, die möglichst einfach sein soll. Die Vorteile sind klar: Der Implementierer kann sich auf »seinen Code« konzentrieren und muß nur an wenigen Stellen auf die Soforthilfe Rücksicht nehmen, indem er Funktionen der Schnittstelle aufruft. Andererseits kann der Gestalter der Soforthilfe durch Kenntnis der Oberfläche und der Eigenschaften einer Applikation unabhängig von der Implementierung durch den Programmierer seine ergonomischen Ideen (sogar parallel dazu) realisieren.

Die Anzeige der Hilfetexte, ihre Auswahl und die Einbindung in eine Applikation sollte also applikationsunabhängig implementiert sein. Man wird daher den erforderlichen Code zur Steuerung und Verwaltung der Hilfetexte in ein eigenständiges Modul legen, auf das der applikations-

spezifische Code zugreift und das mit den applikationsspezifischen Hilfetexten parametrisiert wird. Es ergibt sich somit eine Struktur gemäß *Bild 1*. Die Weiterentwicklung der Soforthilfe wird applikationsübergreifend und läßt sich problemlos auf alle mit ihr realisierten Applikationen übertragen.



► *Bild 1: Aufbau einer Applikation mit der Hilfe-Verwaltung*

Das entwickelte Hilfe-Modul

Das entwickelte Hilfemodul ist eine applikationsunabhängige Implementierung zur Verwaltung von Hilfetexten, die die Fragen des Benutzers nach dem »Wie«, »Was« und »Warum« einer Applikation beantworten sollen. Auch die Kurzbeschreibung von Menübefehlen innerhalb der Statuszeile beim Markieren dieser im Menü gemäß dem IBM-SAA-Standard ist durch das Hilfe-Modul automatisch integriert.

Die Hilfetexte werden unabhängig vom Applikationscode zusammengestellt und verknüpft. Sie können über einen einfachen Index mit Schlüsselwörtern sowie innerhalb von Menüs, Dialog- und Hinweisfeldern mit F1 abgerufen werden. Nicht implementiert wurde das Anklikken von Dialogobjekten mittels des Mauszeigers mit Fragezeichen, wie sie Excel kennt. Es hätte den Rahmen des Artikels gesprengt, ist aber auch nicht so kompliziert, wie es zunächst aussieht. Vielleicht wird in einem späteren Beitrag einmal darauf eingegangen.

Die bei Excel vermißte Zuordnung von Hilfetexten zu einzelnen Elementen eines Dialogfelds wurde dagegen implementiert. Man setzt hierzu den Eingabefokus auf das gewünschte Element und betätigt F1. Im Hilfefenster erscheint der zum dem Eingabeelement passende Hilfetext.

Der Implementierer der Soforthilfe muß zum Zuordnen der Hilfetexte zu Befehlen, Hinweisen etc. der Applikation zwar die Funktionsweise der Applikation genau kennen (beispielsweise aus dem Pflichtenheft), über die Implementierung braucht er dagegen nur verhältnismäßig wenig Einzelheiten zu kennen, beispielsweise wie Befehle sowie Dialogelemente und -felder intern benannt sind. Im wesentlichen beschränkt sich dies auf Informationen, die in der Ressourcen-Datei angegeben sind, insbesondere der Aufbau der Menüs und der Dialogfelder (*dialog boxes*). Da es außer dem Programmcode und der Res-

►► *Listing 1: Ende*

```
Parameters:
 hiNew      is the handle to the new instance of window.
 hiPrev     is the handle to the first instance of window (or NULL).
 rzCmdLine  points to the command line buffer.
 vCmdShow   is the entry style of window for ShowWindow().

Return:
 TRUE is returned if initialization complete,
 otherwise FALSE (memory too small).

*********************************************************************/

BOOL InitInstance
      (HANDLE hiNew,HANDLE hiPrev,LPSTR rzCmdLine,int vCmdShow)

 {BYTE z[80];
  WORD sz;

  hiMain=hiNew; /* set instance global */
  /* —— load the two strings for the no-memory error message —— */
  sz=LoadString(hiMain,STAPPTITLE,z,sizeof(z));
  rzAppTitle=(BYTE*)LocalAlloc(LMEM_FIXED,sz+1);
  if (!rzAppTitle) return FALSE;
  strcpy(rzAppTitle,z);
  sz=LoadString(hiMain,STNOMEM,z,sizeof(z));
  rzNoMemory=(BYTE*)LocalAlloc(LMEM_FIXED,sz+1);
  if (!rzNoMemory) return FALSE;
  strcpy(rzNoMemory,z);
  /* ——— initialize first/further instance ——— */
  if (!hiPrev)
   {/* first instance: create window class, exit if error */
    if (!CreateClass()) goto MemError;
   } /* if */
  /* create global application window */
  if (!CreateAppWindow()) goto MemError;
  /* load accelerator table */
  hAccel=LoadAccelerators(hiMain,MAKEINTRESOURCE(ACC_MAIN));
  if (hAccel==NULL) return FALSE;
  /* initialize help manager */
  HpmInit(hwMain,hiMain);
  /* show created main window */
  ShowWindow(hwMain,vCmdShow); UpdateWindow(hwMain);
  return TRUE;
  MemError:
  /* error: memory too small */
  ErrorNoMem(NULL);
  return FALSE; /* return with error */
 } /* InitInstance() */


/*********************************************************************

 ——————————————————— Main function ———————————————————

*********************************************************************/

WORD PASCAL WinMain
      (HANDLE hiNew,HANDLE hiPrev,LPSTR rzCmdLine,int vCmdShow)

 {MSG wm;

  /* Initialize (window and) instance, return if error */
  if (!InitInstance(hiNew,hiPrev,rzCmdLine,vCmdShow))
    return 255;
  /* —— application execution loop —— */
  while (GetMessage(&wm,NULL,0,0))
   {if (!HpmCheckMessage(&wm))
     {/* message for application */
      if (!TranslateAccelerator(hwMain,hAccel,&wm))
       {TranslateMessage(&wm); DispatchMessage(&wm);
       } /* if */
     } /* if */
   } /* while */
  return wm.wParam;
 } /* WinMain() */

/* ==============================
   End of WINDOWS Application SHAPE
   ==============================
*/
```

►► *Listing 2: Die MAKE-Datei SHAPE.MD*

```
CC=CL >$*.ERR -c -AS -Gsw -Od -W2 -Zip $*.C

SHAPE.RES: SHAPE.RC DEFS.H HELP.TXT
 SPLIT # create *.HTX files from HELP.TXT
 RC -r $*
 DEL *.HTX


HELPMGR.OBJ: HELPMGR.C HELPMGR.H
 $(CC)

SHAPE.OBJ: SHAPE.C DEFS.H HELPMGR.H
 $(CC)

SHAPE.EXE: HELPMGR.OBJ SHAPE.OBJ SHAPE.RES SHAPE.DEF
 LINK HELPMGR+SHAPE/A:16,SHAPE/CODEVIEW,,SLIBW+SLIBCEW/NOD,$*;
 RC $*.RES
```

```
/*********************************************************************
 —— S H A P E ———————— MS-Windows Application ———————————
                        Resource file
======================================================================
Copyright 1989 by
         Marcellus Buchheit, Buchheit software research
         Zaehringerstrasse 47, D-7500 Karlsruhe 1
         Phone (0) 721/37 67 76    (West Germany)
All rights reserved —— Release 1.00 of 89-Jul-13
*********************************************************************/

#include <STYLE.H>
#include <HELPMGR.H>
#include "DEFS.H"

STRINGTABLE BEGIN
  STAPPTITLE,"SHAPE-Programm mit Hilfe"
  STNOMEM,"Zu wenig Speicher!\012Bitte beenden Sie eine Applikation."
  STE_NUMVAL,"Illegaler numerischer Wert."
END /* STRINGTABLE */

/* ——————————————————————————————
   application's menu specification
   ——————————————————————————————
*/
MNU_MAIN MENU BEGIN
  POPUP "&Zeichnen" BEGIN
    MENUITEM "&Ellipse\t^E",CMD_ELLIPSE
    MENUITEM "&Raute\t^R",CMD_RHOMBUS
    MENUITEM "&Sector\t^S",CMD_SECTOR
    MENUITEM SEPARATOR
    MENUITEM "&Ende",CMD_EXIT
  END
  POPUP "&Optionen" BEGIN
    MENUITEM "&Gr÷■e...",CMD_SIZE
    MENUITEM "&Farbe...",CMD_COLOR
    MENUITEM "F&ⁿllmuster...",CMD_BRUSH
  END
  MENUITEM "&L+schen",CMD_CLEAR
  POPUP "\a&Hilfe" HELP BEGIN
    MENUITEM "&Hilfe-Index...\tF1",CMD_HELP_INDEX
    MENUITEM "&Kontexthilfe\tUmsch+F1",CMD_HELP_CONTEXT,GRAYED
  END
END /* MNU_MAIN */

/* ————————————————————————
   accelerators specification
   ————————————————————————
*/
/* note: the NOINVERT option must be set to avoid printing of
   the command description text during the use of an accelerator */
ACC_MAIN ACCELERATORS BEGIN
  "^E",CMD_ELLIPSE,ASCII,NOINVERT
  "^R",CMD_RHOMBUS,ASCII,NOINVERT
  "^S",CMD_SECTOR,ASCII,NOINVERT
  VK_F1,CMD_HELP_INDEX,VIRTKEY,NOINVERT
  VK_F1,CMD_HELP_CONTEXT,VIRTKEY,SHIFT,NOINVERT
END /* ACC_MAIN */

/* command description string table description */
STRINGTABLE BEGIN
  HCS_SYSMENU,"Steuermenⁿ: Fensterdarstellung Σndern"
  HCS_SYSREST,"Normale Fenstergr÷■e wiederherstellen"
  HCS_SYSMOVE,"Fensterposition Σndern"
  HCS_SYSSIZE,"Fenstergr÷■e Σndern"
  HCS_SYSMIN, "Fenster-Sinnbild anzeigen"
  HCS_SYSMAX, "Fenster maximal vergr÷■ern"
  HCS_SYSCLOSE,"Fenster schlie■en, SHAPE beenden"
  HCS_POPUP1,"Objekt mit gewΣhlten Optionen zeichnen"
  HCS_ELLIPSE,"Ellipse mit gewΣhlten Optionen zeichnen"
  HCS_RHOMBUS,"Raute mit gewΣhlten Optionen zeichnen"
  HCS_SECTOR,"Halbkreis mit gewΣhlten Optionen zeichnen"
  HCS_EXIT,"SHAPE-Programm beenden"
  HCS_POPUP2,"Optionen zum Zeichnen angeben"
  HCS_SIZE,"Objektgr÷■e angeben"
  HCS_COLOR,"Objektfarbe angeben"
  HCS_BRUSH,"Objekt-Fⁿllmuster angeben"
  HCS_CLEAR,"Gezeichnetes Objekt l+schen"
  HCS_POPHELP,"Hilfe aufrufen"
  HCS_HELP_INDEX,"Stichw+rter fⁿr Hilfe aufrufen"
  HCS_HELP_CONTEXT,"Kontext-Hilfe aufrufen"
END /* STRINGTABLE */

/* ———————————————
   help resources
   ———————————————*/
/* General help entries.
   This list contains the order of the help text */
#define HTX_APPL     3100 /* must be the value of HTX_FIRST */
#define HTX_OBJECT   3101
#define HTX_MNUDRAW  3102
#define HTX_CMDELIP  3103
#define HTX_CMDRHOMB 3104
#define HTX_CMDSECT  3105
#define HTX_CMDEND   3106
#define HTX_OPTION   3107
#define HTX_MNUOPT   3108
#define HTX_SIZE     3109
#define HTX_CMDSIZE  3110
#define HTX_COLOR    3111
#define HTX_CMDCOLOR 3112
#define HTX_FILL     3113
```

◄◄ *Listing 3: Die Ressourcen-Datei SHAPE.RC*

sourcen-Datei keine weiteren Teile gibt, die in die Applikationsdateien übernommen werden können, wird auch die komplette Implementierung der Fehlertexte, der Beschreibung der Struktur und die Art der Anbindung an Befehle, Dialogelemente etc. in der Ressourcen-Datei erfolgen. Die Hilfetexte werden dabei in neuen Ressourcen-Typen abgelegt. Im nächsten Abschnitt wird darauf eingegangen. Im Gegensatz zu Excel und anderen Applikationen existieren also keine weiteren Dateien mit den Hilfetexten wie etwa EXCELHLP.HLP; der Anwender weiß, daß die Applikation nur aus einer Datei besteht, und der Implementierer braucht nicht auf externe Dateien zuzugreifen (mit dem ganzen Anhängsel an Fehlerbehandlungen). Die Hilfetexte in Ressourcen zu legen, hat jedoch auch Vorteile für die Laufzeit. Dies wird im nächsten Abschnitt untersucht.

Hilfetexte als Ressourcen

Ressourcen sind allgemein die bei Windows vorzuziehende Implementierung von Datenobjekten, die beim Aufruf der Applikation bereits vorhanden sind und meist auch nicht geändert werden müssen. Sie können nicht nur die vordefinierten Typen (Zeichenketten, Sinnbilder, Dialogfelder etc.) umfassen, sondern auch beliebige Daten. Viele Programmierer legen jedoch solche Daten immer noch als statische Variablen in ihrem C-Programm an. Diese Variante kostet jedoch viel Speicherplatz, da dadurch das Datensegment DGROUP, auf welches das DS-Segmentregister des Prozessors zeigt, belegt wird. Dieses Datensegment kann nicht ausgelagert und auch häufig nicht in den EMS-Speicher geschoben werden, so daß der Hauptspeicher, das am meisten von Windows benutzte Betriebsmittel, stark belastet wird und somit die Gesamtleistung des gesamten Systems abnimmt. Außerdem werden dieselben Daten für jede Instanz einer Applikation neu angelegt. Ressourcen verhalten sich dagegen exakt wie Programmcode. Bei entsprechender Kennzeichnung in der Ressourcen-Datei werden sie aus dem Speicher entfernt (*discarded*), sobald andere wichtiger benötigte Daten geladen werden und der Speicher zu klein wird. Mehrere Instanzen einer Applikation teilen sich gemeinsame Ressourcen, sie werden also nur einmal geladen (dadurch ist bei der Veränderung von Ressourcen-Daten größte Vorsicht geboten!)

Einer der größeren Implementierungsfehler ist es, Ressourcen zu laden und deren Inhalt anschließend in selbst angelegten Speicher zu kopieren, was einige Programmierer insbesondere bei Zeichenketten tun. Dadurch geht nämlich der Vorteil der effizienten Speichernutzung von Ressourcen wieder verloren. Leider wurde sogar in der früheren »HELLO«-Applikation des Windows-Toolkit diese »Todsünde« vorgeführt.

▸▸ *Listing 3: Fortsetzung*

Wie bereits oben angedeutet, können Ressourcen nicht nur Zeichenketten, Menüs, Sinnbilder, Dialogfelder etc. sein, sondern ein beliebiges, vom Programmierer entwickeltes Format besitzen. Hierzu muß man der Ressource lediglich einen Typ zuweisen. Bei Windows können Ressourcen numerische und namentliche Typen besitzen. Bei ersteren wird der Typ als eine Zahl zwischen 1 und 32767 angegeben, wobei die Standard-Ressourcen Werte zwischen 1 und 10 besitzen und die weiteren Zahlen bis 255 für zukünftige Erweiterungen reserviert sind. Namentliche Typen bestehen aus einer Zeichenkette, die mit 0 abgeschlossen ist. Der Nachteil von Namentypen ist, daß sie mehr Speicherplatz brauchen und langsamer sind. Außerdem kann man durch Analyse des .EXE-Codes leichter ein fremdes Programm analysieren, da die Namen natürlich die Bedeutung einer Ressource besser wiedergeben als eine Zahl. Wir verwenden daher für eigene Ressourcen-Typen ausschließlich Zahlen ab 256.

Mit einem solchen Ressourcen-Typ können jetzt fast beliebig viele Ressourcen definiert werden, wieder entweder mit einer Zahl (zwischen 1 und 32767) oder einem Namen. Aus obigen Gründen werden auch hier numerische Werte verwendet. Die Daten einer eigenen Ressource müssen in einer DOS-Datei stehen, Byte für Byte exakt wie sie in die EXE-Datei übertragen werden sollen und später vom Windows-Programm gelesen werden. In der Ressourcendatei wird eine Ressource wie folgt definiert:
Ressource-Nr Typ-Nr [Optionen] DOS-Dateiname
Bei den Optionen sollte *LOADONCALL*, *MOVEABLE* und *DISCARDABLE* angegeben sein, sofern die Ressource nicht geändert und dauerhaft im Speicher stehen bleiben muß. Ressource-Nummer und Typ-Nummer wird man natürlich zur besseren Lesbarkeit als Konstanten mit `#define` namentlich benennen. Diese selbstdefinierten Ressourcen eignen sich vor allem für Texte, da diese leicht mit einem Editor als Datei erstellt werden können. Bei Binärdateien ist für die Erstellung dagegen häufig ein Generator erforderlich, der auch erst geschrieben sein will.

Wir verwenden für die Hilfetexte und deren Verwaltung insgesamt drei verschiedene Ressourcen-Typen: Die Hilfetexte selbst sind als Ressource-Typ HELPTEXT (257) abgelegt und der Index ist vom Typ HELPLIST (256). Wichtig ist, daß in beiden Fällen die erstellten Textdateien mit dem EOF-Zeichen 26 (Strg-Z) abgeschlossen werden, sonst erkennt das Hilfe-Modul nicht das Ende der Texte. In der Index-Ressource steht in jeder Zeile ein Indexname. Die Namen müssen alphabetisch vorsortiert sein, so wie sie später im Index-Listenfeld erscheinen.

Bei den Hilfetexten selbst wird die erste Zeile als eine Überschrift verwendet, die bei der Anzeige im Hilfefenster oberhalb des Hilfetextfelds erscheint. Die folgenden Zeilen enthalten den

```
#define HTX_CMDFILL  3114
#define HTX_CLEAR    3115
#define HTX_CMDCLR   3116
#define HTX_HELP     3117
#define HTX_MNUHELP  3118
#define HTX_CMDHPIX  3119
#define HTX_CMDHPCTX 3120

/* special help entries (dialog & message help texts) */
#define HTX_VERT     4000
#define HTX_HORZ     4001
#define HTX_SIZEOK   4002
#define HTX_RED      4010
#define HTX_GREEN    4011
#define HTX_BLUE     4012
#define HTX_COLOROK  4013
#define HTX_FILL1    4020
#define HTX_FILL2    4021
#define HTX_FILL3    4022
#define HTX_FILLOK   4023
#define HTX_CANCEL   4100
#define HTX_VERTVAL  4200
#define HTX_HORZVAL  4201

#define LMO LOADONCALL DISCARDABLE MOVEABLE /* load/memory options */

/* help index keyword list */
HLS_INDEX HELPLIST LMO INDEX.HTX

/* index to help-text index table (alphabetical sorted) */
HLS_INDEX RCDATA LMO BEGIN
 HTX_CMDHPCTX, /* first entry is last available general help-text */
 HTX_APPL,HTX_COLOR,HTX_FILL,HTX_SIZE,HTX_HELP,HTX_CLEAR,HTX_OBJECT,
 HTX_OPTION
END /* HLS_INDEX */

/* table of menu command help texts */
HLS_MENU RCDATA LMO BEGIN
 MNU_DRAW,         HTX_MNUDRAW,
 CMD_ELLIPSE,      HTX_CMDELIP,
 CMD_RHOMBUS,      HTX_CMDRHOMB,
 CMD_SECTOR,       HTX_CMDSECT,
 CMD_EXIT,         HTX_CMDEND,
 MNU_OPTION,       HTX_MNUOPT
 CMD_SIZE,         HTX_CMDSIZE,
 CMD_COLOR,        HTX_CMDCOLOR,
 CMD_BRUSH,        HTX_CMDFILL,
 CMD_CLEAR,        HTX_CMDCLR,
 MNU_HELP,         HTX_MNUHELP,
 CMD_HELP_INDEX,   HTX_CMDHPIX,
 CMD_HELP_CONTEXT,HTX_CMDHPCTX,    .
 0 /* end of table */
END /* HLS_MENU */

/* table of dialog box items help texts */
HLS_DIALOG RCDATA LMO BEGIN
 DBX_SIZE,  IDVERT,HTX_VERT,IDHORZ,HTX_HORZ,IDOK,HTX_SIZEOK,0
 DBX_COLOR, IDRED,HTX_RED,IDGREEN,HTX_GREEN,IDBLUE,HTX_BLUE,
            IDOK,HTX_COLOROK,0
 DBX_BRUSH, IDHOLLOW,HTX_FILL1,IDHATCH,HTX_FILL2,IDSOLID,HTX_FILL3,
            IDOK,HTX_FILLOK,0
         0, IDCANCEL,HTX_CANCEL,0 /* default dialog box */
END /* HLS_DIALOG */

/* table of message box help texts */
HLS_MESSAGE RCDATA LMO BEGIN
 STE_NUMVAL,DBX_SIZE,IDHORZ,HTX_HORZVAL,IDVERT,HTX_VERTVAL,0
            0,0 /* end of default dialog box */
 0 /* end of message string list */
END /* HLS_MESSAGE */

/* help text resources */
HTX_APPL     HELPTEXT LMO APPL.HTX
HTX_OBJECT   HELPTEXT LMO OBJECT.HTX
HTX_MNUDRAW  HELPTEXT LMO MNUDRAW.HTX
HTX_CMDELIP  HELPTEXT LMO CMDELIP.HTX
HTX_CMDRHOMB HELPTEXT LMO CMDRHOMB.HTX
HTX_CMDSECT  HELPTEXT LMO CMDSECT.HTX
HTX_CMDEND   HELPTEXT LMO CMDEND.HTX
HTX_OPTION   HELPTEXT LMO OPTION.HTX
HTX_MNUOPT   HELPTEXT LMO MNUOPT.HTX
HTX_SIZE     HELPTEXT LMO SIZE.HTX
HTX_CMDSIZE  HELPTEXT LMO CMDSIZE.HTX
HTX_COLOR    HELPTEXT LMO COLOR.HTX
HTX_CMDCOLOR HELPTEXT LMO CMDCOLOR.HTX
HTX_FILL     HELPTEXT LMO FILL.HTX
HTX_CMDFILL  HELPTEXT LMO CMDFILL.HTX
HTX_CLEAR    HELPTEXT LMO CLEAR.HTX
HTX_CMDCLR   HELPTEXT LMO CMDCLR.HTX
HTX_HELP     HELPTEXT LMO HELP.HTX
HTX_MNUHELP  HELPTEXT LMO MNUHELP.HTX
HTX_CMDHPIX  HELPTEXT LMO CMDHPIX.HTX
HTX_CMDHPCTX HELPTEXT LMO CMDHPCTX.HTX
HTX_VERT     HELPTEXT LMO VERT.HTX
HTX_HORZ     HELPTEXT LMO HORZ.HTX
HTX_SIZEOK   HELPTEXT LMO SIZEOK.HTX
HTX_RED      HELPTEXT LMO RED.HTX
HTX_GREEN    HELPTEXT LMO GREEN.HTX
HTX_BLUE     HELPTEXT LMO BLUE.HTX
HTX_COLOROK  HELPTEXT LMO COLOROK.HTX
HTX_FILL1    HELPTEXT LMO FILL1.HTX
HTX_FILL2    HELPTEXT LMO FILL2.HTX
```

```
HTX_FILL3    HELPTEXT LMO FILL3.HTX
HTX_FILLOK   HELPTEXT LMO FILLOK.HTX
HTX_CANCEL   HELPTEXT LMO CANCEL.HTX
HTX_VERTVAL  HELPTEXT LMO VERTVAL.HTX
HTX_HORZVAL  HELPTEXT LMO HORZVAL.HTX
/* ───────────────────────
   dialog boxes specification
   ───────────────────────*/
DBX_SIZE DIALOG LOADONCALL MOVEABLE DISCARDABLE 20,20,170,47
CAPTION "Shape: Objektgr■e"
STYLE WS_BORDER|WS_CAPTION|WS_DLGFRAME|WS_POPUP
BEGIN
 CONTROL "&Vertikal:",IDNONE,"static",SS_LEFT|WS_CHILD,28,8,36,8
 CONTROL "",IDVERT,"edit",ES_LEFT|WS_BORDER|WS_GROUP|WS_TABSTOP|
         WS_CHILD,70,6,20,12
 CONTROL "%",IDNONE,"static",SS_LEFT|WS_CHILD,94,8,4,8
 CONTROL "&Horizontal:",IDNONE,"static",SS_LEFT|WS_CHILD,20,30,44,8
 CONTROL "",IDHORZ,"edit",ES_LEFT|WS_BORDER|WS_GROUP|WS_TABSTOP|
         WS_CHILD,70,28,20,12
 CONTROL "%",IDNONE,"static",SS_LEFT|WS_CHILD,94,30,4,8
 CONTROL "Ok",IDOK,"button",BS_DEFPUSHBUTTON|WS_GROUP|WS_TABSTOP|
         WS_CHILD,114,6,50,14
 CONTROL "Abbrechen",IDCANCEL,"button",
         BS_PUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,114,28,50,14
END /* DBX_SIZE */
DBX_COLOR DIALOG LOADONCALL MOVEABLE DISCARDABLE 20,20,144,66
CAPTION "Shape: Objektfarbe"
STYLE WS_BORDER|WS_CAPTION|WS_DLGFRAME|WS_POPUP
BEGIN
 CONTROL "&rot:",IDTRED,"static",SS_LEFT|WS_CHILD,8,8,16,8
 CONTROL "",IDRED,"ScrollBar",SBS_HORZ|WS_TABSTOP|WS_CHILD,30,8,32,8
 CONTROL "0%",IDVRED,"static",SS_RIGHT|WS_CHILD,64,8,16,8
 CONTROL "&grⁿn:",IDTGREEN,"static",SS_LEFT|WS_CHILD,4,30,20,8
 CONTROL "",IDGREEN,"ScrollBar",SBS_HORZ|WS_TABSTOP|WS_CHILD,
         30,30,32,8
 CONTROL "0%",IDVGREEN,"static",SS_RIGHT|WS_CHILD,64,30,16,8
 CONTROL "&blau:",IDTBLUE,"static",SS_LEFT|WS_CHILD,4,52,20,8
 CONTROL "",IDBLUE,"ScrollBar",SBS_HORZ|WS_TABSTOP|WS_CHILD,
         30,52,32,8
 CONTROL "0%",IDVBLUE,"static",SS_RIGHT|WS_CHILD,64,52,16,8
 CONTROL "Ok",IDOK,"button",BS_DEFPUSHBUTTON|WS_GROUP|WS_TABSTOP|
         WS_CHILD,88,28,50,14
 CONTROL "Abbrechen",IDCANCEL,"button",
         BS_PUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,88,48,50,14
END /* DBX_COLOR */
DBX_BRUSH DIALOG LOADONCALL MOVEABLE DISCARDABLE 20,20,148,70
CAPTION "Shape: Fⁿllmuster"
STYLE WS_BORDER|WS_CAPTION|WS_DLGFRAME|WS_POPUP
BEGIN
 CONTROL "",IDNONE,"static",SS_BLACKFRAME|WS_GROUP|WS_CHILD,
         32,14,34,12
 CONTROL "",IDTHATCH,"static",WS_BORDER|SS_USERITEM|WS_GROUP|WS_CHILD,
         32,32,34,12
 CONTROL "",IDTSOLID,"static",WS_BORDER|SS_USERITEM|WS_GROUP|WS_CHILD,
         32,50,34,12
 CONTROL "Muster",IDNONE,"button",BS_GROUPBOX|WS_GROUP|WS_CHILD,
         6,0,66,66
 CONTROL "&1",IDHOLLOW,"button",BS_RADIOBUTTON|WS_GROUP|WS_TABSTOP|
         WS_CHILD,10,14,14,12
 CONTROL "&2",IDHATCH,"button",BS_RADIOBUTTON|WS_TABSTOP|WS_CHILD,
         10,32,14,12
 CONTROL "&3",IDSOLID,"button",BS_RADIOBUTTON|WS_TABSTOP|WS_CHILD,
         10,50,14,12
 CONTROL "Ok",IDOK,"button",BS_DEFPUSHBUTTON|WS_GROUP|WS_TABSTOP|
         WS_CHILD,92,30,50,14
 CONTROL "Abbrechen",IDCANCEL,"button",
          BS_PUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,92,50,50,14
END /* DBX_BRUSH */
/* ===========================
   Resources of the help manager
   =============================*/
DBX_HPMINDEX DIALOG LOADONCALL MOVEABLE DISCARDABLE 50,30,140,112
CAPTION "SHAPE-Hilfe (Index)"
STYLE WS_VISIBLE|WS_BORDER|WS_CAPTION|WS_DLGFRAME|WS_SYSMENU|WS_POPUP
BEGIN
 CONTROL "Hilfe-&Index",IDNONE,"static",SS_LEFT|WS_GROUP|WS_CHILD,
         6,6,44,8
 CONTROL "",IDLIST,"listbox",LBS_NOTIFY|WS_VSCROLL|WS_BORDER|WS_GROUP|
         WS_TABSTOP|WS_CHILD,6,18,128,65
 CONTROL "Hilfe",IDOK,"button",BS_DEFPUSHBUTTON|WS_GROUP|WS_TABSTOP|
         WS_CHILD,6,92,50,14
 CONTROL "Beenden",IDCANCEL,"button",BS_PUSHBUTTON|WS_TABSTOP|
         WS_CHILD,84,92,50,14
END /* DBX_HPMINDEX */

DBX_HPMTEXT DIALOG LOADONCALL MOVEABLE DISCARDABLE 50,30,226,122
CAPTION "SHAPE-Hilfe"
STYLE WS_VISIBLE|WS_BORDER|WS_CAPTION|WS_DLGFRAME|WS_SYSMENU|WS_POPUP
BEGIN
 CONTROL "",IDTITLE,"static",SS_LEFT|WS_CHILD,4,4,214,8
 CONTROL "",IDTEXT,"edit",ES_MULTILINE|WS_BORDER|WS_GROUP|WS_CHILD,
         4,14,210,86
 CONTROL "",IDTXSRL,"ScrollBar",SBS_VERT|WS_GROUP|WS_TABSTOP|WS_CHILD,
         214,14,8,86
 CONTROL "&Index",IDOK,"button",
         BS_DEFPUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,4,104,50,14
 CONTROL "&Beenden",IDCANCEL,"button",
         BS_PUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,60,104,50,14
 CONTROL "&Zurⁿck",IDPREV,"button",
         BS_PUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,116,104,50,14
 CONTROL "&Weiter",IDNEXT,"button",
         BS_PUSHBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,172,104,50,14
END /* DBX_HPMTEXT */
```

◂◂ *Listing 3: Fortsetzung*

Hilfetext, wobei eine neue Zeile mit Return eingeleitet wird. Das Hilfemodul bewirkt einen automatischen wortweisen Zeilenumbruch, so daß die Formatierung nicht sehr kritisch ist. Im Beispiel ist die Breite des Hilfetextes auf 50 Zeilen beschränkt. Wichtig ist auch hier, bei der Eingabe nicht das EOF-Zeichen zu vergessen.

Die Hilfetexte müssen mit dem Ressourcen-Compiler einzeln in die Applikationsdatei eingebaut werden, wofür sie auch als einzelne Dateien vorliegen müssen. Um nun zu verhindern, daß eine Vielzahl kleinster Textdateien angelegt und bearbeitet werden müssen, wird eine zusammenhängende Hilfetext-Datei mit Namen HELP.DAT verwendet, die alle Hilfetexte enthält. Für die Verarbeitung der Hilfetexte durch den Ressourcen-Compiler wird diese Datei automatisch von einem kleinen Hilfsprogramm mit Namen SPLIT zerlegt (siehe unten). Dieses Programm fügt auch die erforderlichen EOF-Zeichen am Ende dieser kleinen Dateien an.

Da die Eingabe der Umlaute und Sonderzeichen außerhalb von Windows nicht ganz einfach ist, ist es empfehlenswert, die Hilfetexte mit dem Notiz-Programm (oder bei größeren Text mit MS-Write) unter Windows zu erstellen. So kann man auch gleichzeitig die Applikation, für welches die Hilfetexte geschrieben werden müssen, testen und inspizieren.

Die Zuordnung von Hilfetexten zu Menübefehlen, Dialogfeldern, Hinweiszeichenketten etc. erfolgt mittels Konstanten-Bezeichnern in festen Tabellen. Für die Realisierung dieser Tabellen bietet sich der vordefinierte Ressourcen-Typ »Row-Data« als dritter verwendeter Typ an. Er speichert angegebene Daten als 16-Bit-Werte nacheinander ab. Wie die Verwaltungstabellen damit genau aufgebaut werden, wird im nächsten Abschnitt erklärt.

Die Daten-Schnittstelle des Hilfemoduls

Die applikationsspezifischen Angaben für die Hilfeverwaltung werden alle in der Ressourcen-Datei der Applikation untergebracht. Es ist sinnvoll, die folgenden Erläuterungen anhand der Ressourcen-Datei und der Definitionsdatei der Beispielapplikation in *Listing 3* und *4* zu verfolgen. Die applikationsspezifischen Erläuterungen dieser Listings erfolgen allerdings erst weiter unten.

Die Zeichenketten für die Menü-Kurzbeschreibungen, die in der Statuszeile erscheinen, werden innerhalb einer STRINGTABLE vereinbart. Die Zeichenketten erhalten dabei Werte, die sich exakt um den Wert 2000 von den Befehlscodes der Menüeinträge unterscheiden. Der Übersicht halber sollten diese Zeichenketten-Werte mit »HCS_« beginnen. Die Befehlscodes werden in der Definitionsdatei wie üblich festgelegt. Eine

Aufschlagmenü-Überschrift (in der Menüleiste) besitzt aber keinen Befehlscode. Ihr wird deshalb als Pseudo-Code der Wert des ersten Befehlscodes des Menüs um eins erniedrigt zugewiesen. Somit kann auch diesem Eintrag eine Zeichenkette zugewiesen werden. Leider kann der Ressourcen-Editor bei `#define`-Angaben nicht mit Arithmetik umgehen. Angaben wie

```
#define HCS_ELLIPSE CMD_ELLIPSE+2000
```

werden mit »Syntax-Error« an der Einfügeposition von HCS_ELLIPSE beantwortet. So muß man Konstanten für die Statuszeilen-Zeichenketten neu vereinbaren. Bei Veränderungen von Menüs hält sich der Aufwand in Grenzen, wenn die Befehlscodes in neuen Menüs mit gewissen Abständen beginnen, im Beispiel wurde hierfür die Zahl 20 gewählt. Da sich die Angaben auf die Befehlscodes der Menüs beziehen, sind problemlos dynamische Änderungen in den Menüs durch die Applikation möglich, sofern die Menübefehle eindeutige Befehlscodes erhalten. In der Zeichenketten-Tabelle sollten auch für die vordefinierten Menübefehle des Systemmenüs Beschreibungen angegeben werden. Der allgemeine Aufbau dieser Ressource sollte den Angaben in *Tabelle 1* entsprechen. Die angegebenen Konstanten sind in der Kopfdatei der Hilfeverwaltung `HELPMGR.H` (*Listing 6*) vereinbart.

▸ *Tabelle 1: Aufbau der Zeichenkettentabelle für die Statuszeilen-Kurzbeschreibung.*

STRINGTABLE BEGIN
/* Beschreibung des Steuermenüs */
 HCS_SYSMENU, *Zeichenkette für Steuermenü*
 HCS_SYSREST, *Zeichenkette für Befehl »Wiederherstellen«*
 HCS_SYSMOVE, *Zeichenkette für Befehl »Bewegen«*
 HCS_SYSSIZE, *Zeichenkette für Befehl »Größe ändern«*
 HCS_SYSMIN, *Zeichenkette für Befehl »Sinnbild«*
 HCS_SYSMAX, *Zeichenkette für Befehl »Vollbild«*
 HCS_SYSCLOSE, *Zeichenkette für Befehl »Schließen«*
/* Beschreibung des 1. Aufklappmenüs */
 Code des 1.Menübefehls+1999, Beschreibung des Menüs
 Code des 1.Menübefehls+2000, Beschreibung des Befehls
 ...
 Code des letzten Menübefehls+2000, Beschreibung des Befehls
/* Beschreibung des 2. Aufklappmenüs */
 Code des 1.Menübefehls+1999, Beschreibung des Menüs
 ...
/* Beschreibung des Hilfe-Menüs */
 HCS_POPHELP, *Zeichenkette für Hilfe-Menü*
 HCS_HELP_INDEX, *Zeichenkette für »Hilfe-Index«*
 HCS_HELP_CONTEXT, *Zeichenkette für »Kontexthilfe«*
END /* STRINGTABLE */

Hilfetexte werden mit Nummern versehen. Allgemein sollten diese als Konstanten mit `#define` vereinbart werden und mit »HTX_« beginnen. Ihre Werte müssen mit dem Wert der Konstante HTX_FIRST (3000) beginnen und zunächst fortlaufend sein. Diese zusammenhängenden Texte geben in ihrer Reihenfolge die Liste der allgemeinen Hilfetexte an. Die Hilfeverwaltung erkennt sie als eine Einheit. Üblicherweise verweisen die Einträge des Hilfe-Indexes auf diese Liste. Werte ab der Konstante HTX_SPECIAL (4000) bezeichnen dagegen spezielle Hilfetexte, die nicht in der allgemeinen Textliste vertreten sind. Sie werden beispielsweise bei der Beschreibung von Dialogelement-Einträgen angesprochen. In der Ressourcen-Datei müssen die

▸▸ *Listing 4: Die Konstanten-Definitionsdatei DEFS.H*

```
/* menu commands + popup-entries */
#define MNU_DRAW    120
#define CMD_ELLIPSE 121
#define CMD_RHOMBUS 122
#define CMD_SECTOR  123
#define CMD_EXIT    124
#define MNU_OPTION  140
#define CMD_SIZE    141
#define CMD_COLOR   142
#define CMD_BRUSH   143
#define CMD_CLEAR   160

/* dialog boxes */
#define DBX_SIZE 500
#define DBX_COLOR 501
#define DBX_BRUSH 502

/* main menu */
#define MNU_MAIN 900

/* accelerator */
#define ACC_MAIN 900

/* strings */
#define STAPPTITLE    1000
#define STNOMEM       1001

/* error strings */
#define STE_NUMVAL    1500

/* command description strings */
#define HCS_POPUP1     2120
#define HCS_ELLIPSE    2121
#define HCS_RHOMBUS    2122
#define HCS_SECTOR     2123
#define HCS_EXIT       2124
#define HCS_POPUP2     2140
#define HCS_SIZE       2141
#define HCS_COLOR      2142
#define HCS_BRUSH      2143
#define HCS_CLEAR      2160

/* dialog box entries */
#define IDNONE    0
#define IDHORZ    4000
#define IDVERT    4001
#define IDRED     4010
#define IDGREEN   4011
#define IDBLUE    4012
#define IDTRED    4013
#define IDTGREEN  4014
#define IDTBLUE   4015
#define IDVRED    4016
#define IDVGREEN  4017
#define IDVBLUE   4018
#define IDHOLLOW  4020
#define IDHATCH   4021
#define IDSOLID   4022
#define IDTHATCH  4023
#define IDTSOLID  4024
```

▸▸ *Listing 5: Die Linker-Definitionsdatei SHAPE.DEF*

```
NAME SHAPE
REALMODE
EXETYPE WINDOWS
DESCRIPTION 'Shape program with help manager'
STUB 'WINSTUB.EXE'
CODE MOVABLE DISCARDABLE SHARED
DATA MOVABLE MULTIPLE
HEAPSIZE  4096
STACKSIZE 4096

EXPORTS
 fwMain @1
 fdSize @2
 fdColor @3
 fdBrush @4
; help manager functions
 fdHelpIndex @5
 fdHelpText  @6
 fwSubEdit   @7
 fhMsgFilter @8
```

Werte für alle Hilfetexte von Hand angegeben werden. Leider erlaubt der Ressourcen-Compiler keine flexiblere Beschreibung der Werte.

Es folgen fünf Ressourcen, die alle Verwaltungsdaten für die Soforthilfe beinhalten. Die erste ist vom selbstdefinierten Ressource-Typ HELPLIST und besitzt den Namen HLS_INDEX. Sie enthält alphabetisch sortiert die Schlüsselwörter des Hilfe-Indexes. Sie werden in der Datei

```
/* private resource types */
#define HELPLIST 256
#define HELPTEXT 257

/* commands for help menu */
#define MNU_HELP 280
#define CMD_HELP_INDEX 281
#define CMD_HELP_CONTEXT 282

/* constants for sys-menu description */
#define HCS_SYSMENU      2100
#define HCS_SYSREST      2101
#define HCS_SYSMOVE      2102
#define HCS_SYSSIZE      2103
#define HCS_SYSMIN       2104
#define HCS_SYSMAX       2105
#define HCS_SYSCLOSE     2106
#define HCS_POPHELP      2280
#define HCS_HELP_INDEX   2281
#define HCS_HELP_CONTEXT 2282

/* entry functions of the help manager */
BOOL HpmInit(HWND hwAppl,HWND hiAppl);
BOOL HpmCheckMessage(LPMSG rwm);
LONG HpmProcessHelpMsg(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2);
VOID HpmSetCmdLine(WORD syClient);
VOID HpmSetMsgBoxEnv(WORD iItem,WORD iString);
VOID HpmSetDlgBoxEnv(WORD iDlgBox);

/* constants for help dialog boxes */
#define DBX_HPMINDEX 3000
#define DBX_HPMTEXT  3001

/* list of help index names */
#define HLS_INDEX 3030
#define HLS_MENU 3031
#define HLS_DIALOG 3032
#define HLS_MESSAGE 3033

/* help text areas
   HTX_FIRST..HTX_SPECIAL-1 is sorted by chapters, enter by index */
#define HTX_FIRST   3100
#define HTX_SPECIAL 4000

/* constants for help dialog items (can be used by applicaion too) */
#define IDLIST  3200
#define IDTEXT  3201
#define IDTITLE 3202
#define IDINDEX 3203
#define IDPREV  3204
#define IDNEXT  3205
#define IDTXSRL 3206
```

◂◂ *Listing 6: Die Kopfdatei HELPMGR.H der Hilfeverwaltung.*

INDEX.HTX angegeben und vom Compiler als Ressource geladen. Anschließend folgt die Zuordnungstabelle des Hilfe-Indexes. Sie ist vom Typ »Row Data«, besitzt den Namen HLS_INDEX und enthält zunächst den Wert des letzten Hilfetextes in der allgemeinen Textliste. Er muß angegeben werden, damit die Hilfeverwaltung weiß, wann die Liste zu Ende ist. Im folgenden wird für jeden der alphabetisch sortierten Einträge in der Liste angegeben, welcher Hilfetext referenziert werden soll, wenn der Benutzer den Index-Eintrag auswählt. Im Beispiel heißt der erste Eintrag »Applikation«, entsprechend wird der Hilfetext mit dem Index HTX_APPL angesprochen, der in der Datei APPL.HTX steht und die Applikation allgemein beschreibt. *Tabelle 2* beschreibt das allgemeine Format der HLS_INDEX-Ressource.

```
HLS_INDEX RCDATA LOADONCALL DISCARDABLE MOVEABLE
  Nummer des letzten Textes in der allgemeinen Hilfetextliste
  Textnummer für das erste Index-Schlüsselwort
  Textnummer für das zweite Index-Schlüsselwort
  ...
  Textnummer für das letzten Schlüsselwort im Index
END /* HLS_INDEX */
```

◂◂ *Tabelle 2: Aufbau der Tabelle HLS_INDEX für die Verwaltung des Hilfe-Indexes.*

Die nächsten Ressourcen dienen der Zuordnung von Hilfetexten an Menübefehle, Dialog-Einträge und Hinweisfelder. Sie sind alle vom vordefinierten Typ »Row-Data«. Ihr Aufbau ist in *Tabelle 3* bis *Tabelle 5* allgemein beschrieben. Die Tabelle HLS_MENU wird verwendet, wenn innerhalb eines aufgeschlagenen Menüs F1 gedrückt wird. Sie sucht dann innerhalb dieser Tabelle den für den selektierten Befehl passenden Text. Auch hier gilt wieder für die Überschriften der Aufklapp-Menüs, daß sie einen Wert besitzen, der um 1 kleiner ist als der Befehlscode des ersten Menüeintrags (etwa MNU_OPTION = CMD_SIZE - 1). Es brauchen nicht alle Befehle mit Hilfetexten versehen zu werden. Fehlt ein Befehl in der Tabelle, wird F1 ignoriert.

```
HLS_MENU RCDATA LOADONCALL DISCARDABLE MOVEABLE
/* Hilfetexte für das 1. Aufklappmenü */
  Code des 1.Menübefehls-1, Textnummer für das Menüs
  Code des 1.Menübefehls, Textnummer für den 1.Befehl
  ...
  Code des letzten Menübefehls, Textnummer für den letzten Befehl
/* Hilfetexte für das 2. Aufklappmenü */
  Code des 1.Menübefehls-1, Textnummer für das Menü
  ...
/* Hilfetexte für Beschreibung des Hilfe-Menüs */
  MNU_HELP, Textnummer für Hilfe-Menü
  CMD_HELP_INDEX, Textnummer für »Hilfe-Index«
  CMD_HELP_CONTEXT, Textnummer für »Kontexthilfe«
  0 /* Ende der Tabelle */
END /* HLS_MENU */
```

◂ *Tabelle 3: Aufbau der Tabelle HLS_MENU für die Verwaltung der Menü-Hilfetexte.*

```
HLS_DIALOG RCDATA LOADONCALL DISCARDABLE MOVEABLE
/* Hilfetexte für Einträge des ersten Dialogfelds */
  Dialogfeldnummer,
  Nummer 1.Eintrag, Textnummer, ..., Nummer letzter Eintrag,
         Textnummer
  0 /* Ende der Untertabelle für erstes Dialogfeld */
/* Hilfetexte für Einträge des zweiten Dialogfelds */
  Dialogfeldnummer,
  ...
  0
/* Hilfetexte für Einträge in allen Dialogfeldern */
  0 /* Erkennung daß letzte Untertabelle */
  Nummer 1.Eintrag, Textnummer, ..., Nummer letzter Eintrag,
         Textnummer
  0 /* Ende der gesamten Tabelle */
END /* HLS_DIALOG */
```

◂ *Tabelle 4: Aufbau der Tabelle HLS_DIALOG für die Verwaltung der Dialogfeld-Hilfetexte.*

```
HLS_MESSAGE RCDATA LOADONCALL DISCARDABLE MOVEABLE
/* Hilfetexte für den ersten Hinweisfeld-Text */
  Zeichenketten-Nummer des ersten Hinweisfeld-Texts
  Tabelle der Dialogfelder /* Aufbau wie Tabelle 4 */
/* Hilfetexte für zweiten Hinweisfeld-Text */
  Zeichenketten-Nummer des zweiten Hinweisfeld-Texts
..
/* Hilfetexte für den letzten Hinweisfeld-Text */
  Zeichenketten-Nummer des letzten Hinweisfeld-Texts
  Tabelle der Dialogfelder /* Aufbau wie Tabelle 4 */
  0 /* Ende der Tabelle */
END /* HLS_MESSAGE */
```

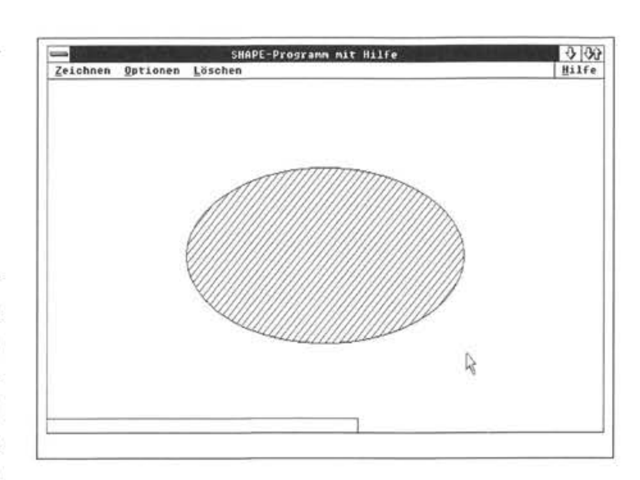
◂ *Tabelle 5: Aufbau der Tabelle HLS_MESSAGE für die Verwaltung der Hinweisfelder-Hilfetexte.*

Die Ressource HLS_DIALOG ordnet den Einträgen in einem Dialogfeld Hilfetexte zu. Als Kennzeichen werden die Identifikationen der Dialogfelder (Anfang »ID«) genommen. Da jedoch häufig in unterschiedlichen Dialogfeldern Einträge mit gleichen Identifikationen verwendet werden (Begründung siehe weiter unten), wird zur eindeutigen Zuordnung auch die Identifikation des Dialogfelds angegeben. Die Ressource ist damit zweidimensional gegliedert, wie aus *Tabelle 4* hervorgeht: Für jedes Dialogfeld werden die feldspezifischen Einträge und der zuge-

► *Tabelle 6: Die Eintrittsfunktionen der Hilfeverwaltung.*

►► *Bild 2: Die Windows-Applikation SHAPE.*

HpmInit(HWND hwAppl, HWND hiAppl)
HpmCheckMessage(LPMSG rwm)
HpmProcessHelpMsg(HWND hw, WORD iMsg, WORD uP1, DWORD ulP2)
HpmSetCmdLine(WORD syClient)
HpmSetMsgBoxEnv(WORD iItem, WORD iString)
HpmSetDlgBoxEnv(WORD iDlgBox)



ordnete Hilfetext angegeben. Abschließend kann in einem »allgemeinen« Dialogfeld (Angabe 0) ein Eintrag auf einen allgemeinen Hilfetext weisen, der sich nicht auf ein konkretes Dialogfeld bezieht und als Ersatzwert verwendet wird, wenn von der Hilfeverwaltung der Eintrag nicht in den vorangegangen dialogfeldspezifischen Unterlisten gefunden wurde. Im Beispiel etwa wurde das Feld »OK« (IDOK) feldspezifisch angegeben, das Feld »Abbrechen« (IDCANCEL) dagegen allgemein. Die Hilfetexte unterscheiden sich vom Inhalt auch entsprechend. Alle Hilfetexte eines Dialogfelds werden von der Hilfeverwaltung zu einer Textliste zusammengefaßt, in der vor- und zurückgeblättert werden kann. Die Textwert-Angaben müssen daher sortiert und fortlaufend sein, außer in der Ersatzwert-Untertabelle.

Die letzte Tabelle HLS_MESSAGE beschreibt die Hilfetext-Zuordnung für Hinweisfelder. Hierbei wird zunächst von der Identifikation der Zeichenkette ausgegangen, die in dem Hinweisfeld angezeigt wurde. Um einen Bezug zur Ursache des Hinweises herzuleiten, enthält jede Zeichenketten-Angabe in dieser Tabelle eine Liste von Dialogfeldern und deren Einträgen, die im Aufbau exakt der HLS_DIALOG-Ressource entspricht. Die Ressource ist damit dreidimensional organisiert wie *Tabelle 5* zeigt. Es brauchen aber nur *die* Einträge angegeben zu werden, von denen aus der Hinweis erfolgen kann. Damit bleibt die Tabellengröße meistens überschaubar.

Abschließend erfolgt in einer längeren Tabelle die eigentliche Definition der Hilfetexte als Ressourcen. Sie werden über ihren Textwert (Anfang »HTX_«) angesprochen. Die Hilfetexte selbst stehen in den anschließend angegebenen Dateien mit Endung ».HTX«. Hätte der Resourcen-Compiler die gleichen Makrofähigkeiten wie »C«, würde sich die Gestaltung der Tabelle vereinfachen. Nach der Angabe der Hilfetexte ist die Implementierung der Soforthilfe schon abgeschlossen. Im weiteren Verlauf der Ressourcen-Datei müssen noch die Dialogfelder für die Hilfeverwaltung angegeben werden. Dies ist jedoch weitgehend applikationsunabhängig. Lediglich in den Überschriften der Dialogfelder sollten die Namen der Applikationen angegeben werden.

Die Code-Schnittstelle des Hilfemoduls

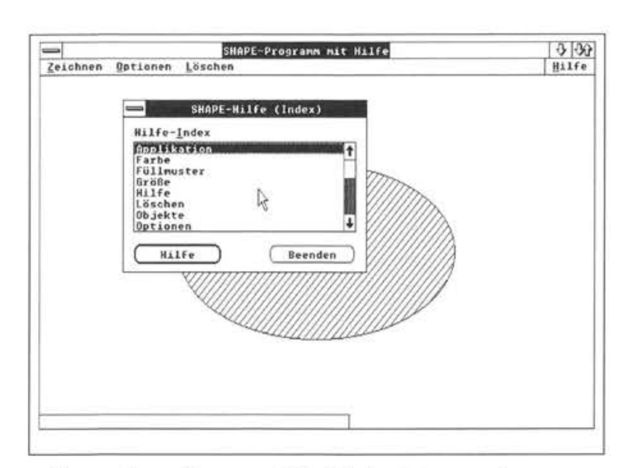
Tabelle 6 beinhaltet alle Funktionen, über die von einer Applikation aus auf das Hilfemodul zugegriffen wird. Die erste Funktion `HpmInit` initialisiert die Hilfe-Verwaltung, legt das Fenster für die Statuszeile an und erzeugt alle weiteren erforderlichen Daten. Sie sollte während der Initialisierung der Applikation aufgerufen werden, aber erst nachdem das Applikationsfenster angelegt worden ist. Als Parameter werden die Bezüge (*handles*) auf die Applikationsinstanz und Applikationsfenster übergeben.

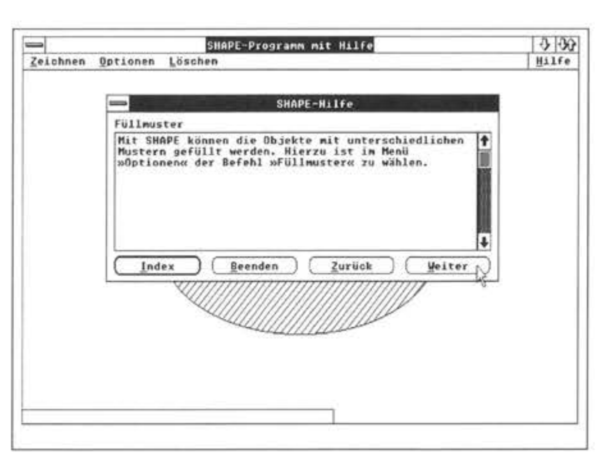
Die Funktion `HpmCheckMessage` überprüft alle Nachrichten, die an die Applikation geschickt werden. Ihr muß nach jedem Aufruf von `GetMessage` in der Nachrichtenschleife der Applikation die empfangene Nachricht übergeben werden. Kehrt die Funktion mit einem Wert ungleich 0 zurück, war die Nachricht für die Hilfeverwaltung bestimmt und darf nicht mehr weiter an `TranslateMessage` etc. übergeben werden.

Auch innerhalb der Applikations-Hauptfensterfunktion müssen einer Modulfunktion alle Nachrichten übergeben werden, die das Hilfemodul gegebenenfalls auswertet. Diese Funktion heißt `HpmProcessHelpMsg` und sollte kurz vor dem Aufruf der Standard-Fensterfunktion `DefWindowProc` für alle Nachrichten aufgerufen werden, die von der Applikation nicht explizit verarbeitet wurden. Die Funktion liefert TRUE zurück, wenn sie die Nachricht so weit verarbeitet hat, daß diese nicht mehr an `DefWindowProc` übergeben werden darf. Andernfalls ist die Nachricht an letztere Funktion wie üblich weiterzuleiten.

Die Anordnung der beiden letzten Funktionen innerhalb der Applikation erinnert an die MDI-Verwaltung von Kevin Welch *[2]*. Auch hier haben im wesentlichen zwei Funktionen die entscheidenden Nachrichten abgefangen und damit den größten Teil der Modulkommunikation übernommen. Tatsächlich lassen sich in der Praxis durch Kaskadierung mehrerer Nachrichten-Abfangfunktionen in der geeigneten Reihenfolge nach `GetMessage` in der Nachrichtenschleife und am Ende der Applikations-Fensterfunktion elegant Module in Applikationen einbinden. Die Methode zeigt erneut die Leistungsfähigkeit des Nachrichtenkonzepts von MS-Windows.

Drei weitere Funktionen übergeben geänderte Daten an das Modul. Man könnte diese Daten effizienter über gemeinsame Variablen über-





◄◄ *Bild 3: Der Index der Hilfe-Verwaltung.*

◄ *Bild 4: Ein angezeigter Hilfetext.*

geben, aber die ausschließliche Verwendung von Funktionen zur Datenübergabe an Module hat den Vorteil, daß man das Modul leicht zu einer dynamischen Link-Library (DLL) umwandeln kann, ohne daß die das Modul aufrufenden Applikationen geändert werden müssen.

`HpmSetCmdLine` legt die Y-Position innerhalb des Applikationsfensters fest, an der das Unterfenster der Statuszeile vom Hilfemodul positioniert werden soll. Diese Funktion muß aufgerufen werden, wenn das Applikationsfenster in der Größe verändert wird.

Die beiden Funktionen `HpmSetMsgBoxEnv` und `HpmSetDlgBoxEnv` übergeben aktuelle Werte für die Zuordnung von Fehlertexten an den augenblicklichen Kontext, falls die Taste F1 innerhalb von Dialog- oder Hinweisfeldern aufgerufen wird. Die erste Funktion wird aufgerufen, bevor ein Text in einem Hinweisfeld ausgegeben wird. Als Argumente wird der Wert des zugrundeliegenden Dialogfeld-Eintrags und der Wert der Zeichenkette aus der Ressourcen-Datei angegeben. Die Funktion `HpmSetDlgBoxEnv` übergibt ähnlich die Nummer der aktuell verwendeten Dialogfeld-Definition aus der Ressourcen-Datei. Aus allen drei Werten ist eine eindeutige Zuordnung auf den gegenwärtigen Kontext möglich.

Die Aufrufe der Hilfeverwaltung können leicht an zentralen Stellen angebracht werden. Sie belasten daher den Implementierer der eigentlichen Applikation kaum. Die Hauptarbeit für die Implementierung der Soforthilfe liegt in der Ressourcen-Datei, also außerhalb des oft komplizierten Programmcodes. Da die Hilfe-Daten in der Ressourcendatei darüberhinaus stark objektorientiert sind, ist es zumindest denkbar, diese Daten interaktiv und automatisch zu erzeugen.

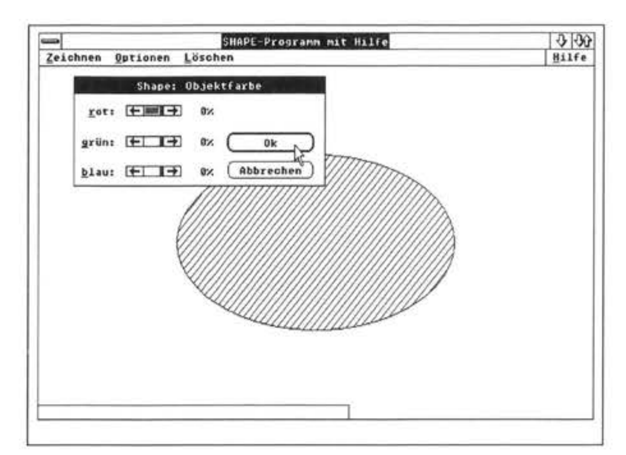
SHAPE: Eine kleine Anwendung des Hilfe-Moduls

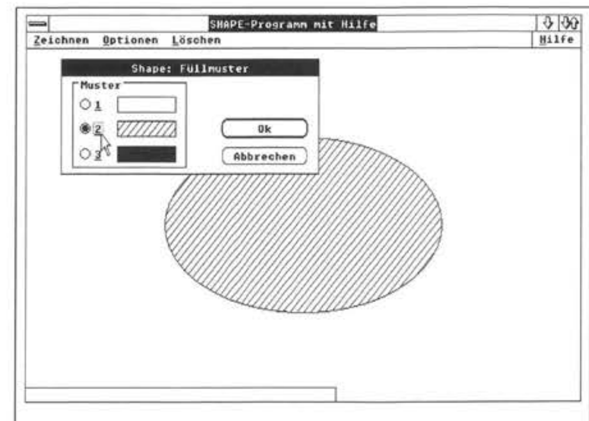
Als Beispiel, wie die Hilfeverwaltung in eine Applikation integriert werden kann, wurde ein kleines Programm zum Zeichnen von einfachen geometrischen Objekten gewählt. Es war mein erstes selbständig entwickeltes und implementiertes Programm für MS-Windows. Ich hatte mich vor längerer Zeit ein wenig in die Beispiele des Windows-Toolkit eingearbeitet und wollte nun eigene Ideen in einer Applikation realisieren, um erste Erfahrungen zu sammeln. Die Ideen waren etwas ausgefallen, es gab außer dem Windows-Toolkit keinerlei Literatur und ich war so unbedarft an die Sache herangegangen, daß ich fast drei Monate benötigte, bis die Applikation endlich fertig war. Erfahrungen hatte ich dabei allerdings reichlich gesammelt. Das Programm heißt SHAPE und kann wahlweise eine Ellipse, eine Raute oder einen halben Ellipsensektor mit wählbarer Farbe und auswählbarem Füllmuster zeichnen, siehe hierzu *Bild 2*. Die Größe des Objekts wird in vertikalen und horizontalen Teilen der Applikationsfenstergröße angegeben. Der Hilfe-Index in *Bild 3* mit den Schlüsselwörtern wird über das Hilfe-Menü oder die Taste F1 aufgerufen. Nachdem ein Begriff ausgewählt worden ist, kann mit Doppel-Klick, der Eingabetaste oder der Schaltfläche »Hilfe« der dazugehörige Text in einem neuen Dialogfeld gelesen werden (*Bild 4*). Der Begriff zeigt dabei in den allgemeinen Teil der Hilfetexte, in dem mit den Schaltflächen »Weiter« und »Zurück« vor und zurück geblättert werden kann. Mit »Index« kann wieder zur Schlüsselwörter-Liste zurückgekehrt werden, »Beenden« schließt die Soforthilfe, ebenso wie die Eingabe von Esc oder ein doppeltes Anklicken des Steuermenüs des Dialogfelds. Die Hilfedarstellung kann angezeigt bleiben, wenn andere Befehle aufgerufen werden. Ein Wechsel zwischen Hilfe-Dialogfeld und Applikation ist mit Alt-F6 möglich.

Die Einstellung der Varianten beim Zeichnen der geometrischen Gebilde erfolgt über drei Dialogfelder im Menü »Optionen«. *Bild 5* zeigt das Dialogfeld zur Einstellung der Farben. Jeder RGB-Farbanteil wird prozentual mit einer Rolleiste eingestellt und als Prozentwert rechts daneben angezeigt. Die Überschriften links von der Rolleiste sind in den betreffenden Farben angezeigt, sofern dies der Bildschirm ermöglicht. Die Auswahl des gewünschten Füllmusters erfolgt mit einem weiteren Dialogfeld. Wie man aus *Bild 6* ersehen kann, werden die Objekte mit dem Füllmuster und mit der eingestellten Farbe im Dialogfeld dargestellt. Weitere Informationen über die Bedienung von SHAPE können aus den Hilfetexten entnommen werden.

▸ *Bild 5: Einstellung der Farben.*

▸▸ *Bild 6: Auswahl des Füllmusters mit Anzeige der Muster.*





Außer über den Index können Hilfetexte auch aus dem aktuellen Kontext aufgerufen werden, indem die Taste F1 betätigt wird. Dies ist beispielsweise aus einem Aufschlagmenü heraus möglich, wenn mit der Tastatur der gewünschte Eintrag markiert, aber nicht aufgerufen wird. Gibt man jetzt F1 ein, erscheint der Hilfetext zum Menü-Befehl (*Bild 7*). Auch innerhalb von Dialogfeldern können (im Gegensatz zu MS-Excel) Hilfetexte zu einzelnen Dialogeinträgen abgerufen werden. Hierzu ist die Eingabemarkierung (der Fokus) auf den gewünschten Eintrag zu setzen. Anschließend ist wieder F1 einzugeben und der entsprechende Text erscheint, siehe *Bild 8*. Ähnlich verhält es sich bei der Hilfeanfrage nach Fehlermeldungen. Wird etwa bei der Einstellung der Objektgröße eine ungültige Zahl oder ein Wert über 100 eingegeben, erscheint zunächst die Fehlermeldung »Illegaler numerischer Wert«. Wird, statt den Hinweis mit »Ok« zu bestätigen, F1 eingegeben, erscheint ein Hilfetext mit der Erläuterung des Fehlers, wobei auch die Bedeutung des Eingabefelds berücksichtigt wird, die zum Fehler beitrug.

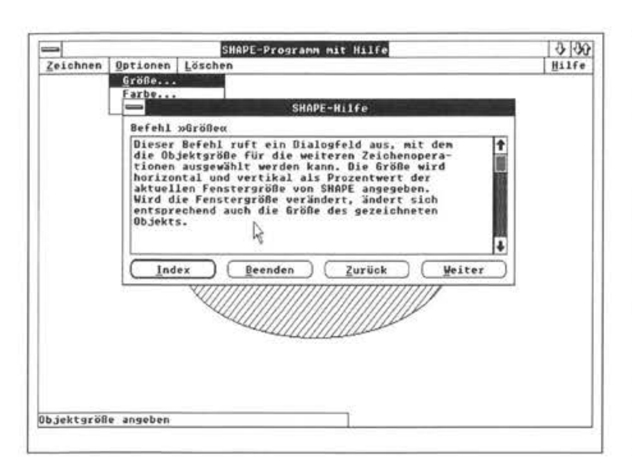
Ein Fehler im Windows-System erlaubt leider nicht die Benutzung der Hilfe-Dialogfenster während andere modale Dialogfenster angezeigt sind. Die Mausbenutzung funktioniert aber weiterhin. Die Ursache dieses Problems wird weiter unten beschrieben.

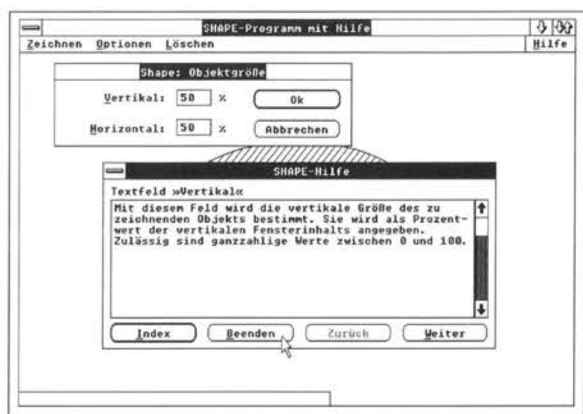
Implementierung von SHAPE

Obwohl SHAPE ziemlich simpel erscheint, zeigt es doch einige recht nützliche Windows-Programmiertechniken. In *Listing 1* ist der C-Quelltext der Applikation abgedruckt. Er greift auf das Modul der Hilfeverwaltung zu, der weiter unten in diesem Artikel erläutert wird. Die MAKE-Datei zum Übersetzen der Quelldateien zeigt *Listing 2*. Da der Ressourcen-Compiler die einzelnen Hilfetexte benötigt, wird die Datei HELP.DAT durch den Aufruf des DOS-Programms SPLIT in einzelne Bestandteile zerlegt. Nach dem Durchlauf des Ressourcen-Compilers werden dann diese Hilfsdateien, die alle mit .HTX enden, wieder gelöscht. Der Ressourcen-Compiler übersetzt die Ressourcen-Datei SHAPE.RC in *Listing 3*. Gemeinsame Definitionen in Ressourcen-Datei und C-Quelldatei sind in DEFS.H in *Listing 4* enthalten. Die Module werden mit LINK, Version 5, zu der Applikation zusammengesetzt. Die hierzu erforderliche Linker-Definitionsdatei ist in *Listing 5* abgedruckt. Wichtig ist, hierin nicht die Fenster- und Dialogfunktionen der Hilfeverwaltung zu vergessen. Die Verbindung zu dem Modul der Hilfeverwaltung wird über dessen Kopfdatei HELPMGR.H (*Listing 6*) erreicht. Die Quellcode der Hilfeverwaltung befindet sich in *Listing 7* als HELPMGR.C. Wir werden weiter unter noch näher darauf eingehen. Alle Hilfetexte sind in der Datei HELP.TXT zusammengefaßt, siehe hierzu *Listing 8*. Jeder einzelne Text beginnt mit einer Zeile, die mit dem Prozentzeichen eingeleitet ist, gefolgt von beliebig weiteren und abschließend dem Namen der .HTX-Zwischendatei. Hierbei muß es sich um den Dateinamen handeln, der entsprechend in der Ressourcen-Datei angegeben ist. Weitere Sonderzeichen existieren in der einfachen Implementierung der Hilfeverwaltung nicht. *Listing 9* zeigt schließlich noch den Quelltext des DOS-Programms SPLIT.C. Neben der Separierung der Datei HELP.TXT fügt es auch die erforderlichen EOF-Zeichen an die Enden der .HTX-Zwischendateien an. Das Programm enthält sonst keine Besonderheiten. Übersetzt wird es durch den Aufruf von »CL SPLIT.C«

Die Ressourcen-Datei SHAPE.RC

Die Ressourcendatei SHAPE.RC ist ziemlich umfangreich, da sie außer der Hilfeverwaltung auch die Dialogfeld-Definitionen des Hilfemoduls umfaßt. In der STRINGTABLE am Anfang sind alle Zeichenketten aufgeführt, die die Applikation zur Ausgabe verwendet. Neben der Applikationsfenster-Überschrift sind dies die Fehlermeldungen. Am Ende der Menüberschreibung ist das Hilfemenü angegeben. Durch Angabe der Option HELP und dem Zeichen »\a« erhält man das charakteristische Erscheinungsbild dieses Eintrags. Nach der Definition der Abkürzungstasten folgen die Daten für die Hilfeverwaltung. Der Aufbau





◂◂ *Bild 7: Hilfe-Aufruf bei Menüs.*

◂ *Bild 8: Hilfe-Aufruf bei Dialogfeldern.*

dieser Ressourcen wurde bereits weiter oben beschrieben. Es wird daher hier nicht mehr weiter darauf eingegangen.

Im folgenden sind die Definitionen der applikationsspezifischen Dialogfelder angegeben. Sie sind weitgehend standardmäßig implementiert. Lediglich bei DBX_BRUSH fällt zweimal der Fensterstil SS_USERITEM auf. Was es damit auf sich hat, wird im nächsten Abschnitt erläutert. Am Schluß folgen die Dialogfenster für die Hilfetext-Anzeige. Man beachte, daß in DBX_HPMTEXT eine zusätzliche Rolleiste neben das Anzeigefeld gelegt wurde, es ist die einzige Möglichkeit, einen rollbaren Text anzuzeigen, ohne daß man diesen verändern kann (siehe weiter unten).

Die Realisierung von SHAPE.C

Nun zu den Besonderheiten innerhalb von SHAPE.C. Bei SHAPE wurde zunächst versucht, prototypenhaft eine einigermaßen »professionelle« Version einer Windows-Applikation zu schreiben. Neben der integrierten Hilfeverwaltung zählen hierzu:

- Verlagerung aller nationaler Texte und Besonderheiten in die Ressourcen-Datei.
- Eine klare Fehlermeldung, wenn der Speicher zu klein ist.
- Vorinitialisierungen von Dialogfeldern und klare Fokus-Positionierungen nach Fehleingaben.
- Verwendung von flexiblen Funktionen für häufig benötigte Aufgaben wie etwa das Einlesen von Daten durch Dialogfelder.

SHAPE eignet sich damit gut als Ausgangspunkt für die Entwicklung neuer Applikationen. Ich schaue mir sehr oft bestimmte Stellen von SHAPE an, um mir wieder ins Gedächnis zu rufen, wie ich ein bestimmtes Problem üblicherweise implementiere. Das betrifft insbesondere die teilweise aufwendige Verwaltung von Daten innerhalb der Dialogfelder. Vereinfachungen ergeben sich etwa in der Art der Funktion ReadDlg–ItemInt: Sie liest nicht nur den Wert des gewünschten Textfelds, sondern überprüft den gelesenen Wert, gibt gegebenenfalls eine Fehlermeldung aus und setzt den Eingabefokus auf das fehlerhafte Feld. In den eigentlichen Dialogfeldfunktionen fdSize ist dann die korrekte und benutzerfreundliche Verarbeitung der numerischen Daten ziemlich einfach. Die Strategie, möglichst viel Datenverwaltung aus Dialogfunktionen in separate Funktionen zu verlagern, wird unterstützt von der Möglichkeit, für die Einträge in den Dialogfeldern feste Konstanten zu wählen. Werden für ähnliche Einträge in unterschiedlichen Dialogfeldern dieselben Konstanten gewählt, kann man ganze Gruppen von Dialogeinträgen durch dieselbe Funktion verarbeiten, als Parameter muß lediglich der Fensterbezug (*window handle*) des Dialogfelds angegeben werden. Bei mehreren gleichartigen Eintragsklassen lassen sich Schleifen für die Verarbeitung der Einträge verwenden. Dies wird in SHAPE anhand der drei Rolleisten für die Farbeinstellung in der Funktion fdColor gezeigt. Nebenbei wird durch unterschiedliche Interpretation der WM_CTLCOLOR-Nachricht erreicht, daß die Texte der Rolleisten in verschiedenen Farben angezeigt werden. Wie man sieht, ist es sehr einfach, durch Abfangen dieser Nachricht farbige Dialogelemente zu erzeugen. Die Fensterfunktion enthält auch die komplette Verarbeitung der WM_HSCROLL-Nachricht einer Rolleiste und ihre sofortige Umsetzung, hier in den entsprechenden Ausgabewert.

Als ich SHAPE implementierte, war es für mich ein größeres Problem, beliebige Bilder in Dialogfeldern zur Verbesserung der Darstellung anzuzeigen. Sinnbilder lassen sich bekanntlich sehr leicht übernehmen, sind aber umständlich zu zeichnen und kennen keine flexiblen Größen. Ein Implementierungsvorschlag für beliebige rechteckige grafische Elemente wird im Dialogfeld DBX_BRUSH für die Auswahl des gewünschten Füllmusters gezeigt: In der Ressourcen-Datei werden die betreffenden Elemente mit der Klasse »static« und dem Stil SS_USERITEM definiert. Wenn nun das Windows-System das Dialogfeld zeichnet, übergibt es der Dialogfunktion, in diesem Falle der Funktion fdBrush, die Nachricht WM_PAINT. Diese kann abgefangen werden, um die gewünschten Elemente selbst zu zeichnen. Dies wird der Übersicht halber in der Funktion PaintDlgItem durchgeführt: Zunächst wird die

▸▸ *Listing 7: Die Datei HELPMGR.C realisiert das Modul für die Hilfeverwaltung.*

Größe des betreffenden Elements ermittelt, der Zeichenkontext (*device context*) angefordert und der bisherige Inhalt des Elements mit `InvalidateRect` ungültig gemacht. Das weitere Zeichnen erfolgt wie sonst üblich: Das gewünschte Füllmuster wird erzeugt, dem Zeichenkontext zugewiesen und das ganze Fenster als Rechteck damit gefüllt. Abschließend werden die erzeugten GDI-Attribute wieder freigegeben.

Die Funktion `ReadDialog` verwaltet das Umfeld beim Erzeugen eines modalen Dialogfelds. Sie liefert TRUE zurück, wenn zum Beenden des Dialogfelds »Ok« eingegeben worden war und FALSE, wenn »Abbrechen« gewählt wurde. Ist zuwenig Speicherplatz vorhanden, um das Dialogfeld aufrufen zu können, wird statt dessen das Hinweisfeld darauf angezeigt. Diese Funktion ruft auch die Funktion `HpmSetDlgBoxEnv` der Hilfe-Verwaltung auf.

In der eigentlichen Fensterfunktion der Applikation mit Namen `fwMain` liegt der Schwerpunkt auf dem Zeichnen des ausgewählten geometrischen Objekts mit den eingestellten Attributen. Am Anfang werden die eingstellten Zeichenattribute (Linienfarbe, Füllmuster) erzeugt. Anschließend wird das gewählte Objekt gezeichnet. Bei der Raute erfolgt dies mit der Windows-Funktion `Polygon`, die einen geschlossenen, füllbaren Kurvenzug erzeugt. Komplizierter gestaltet sich die Zeichnung des halben Ellipsensektors: Man erzeugt zunächst eine Windows-Region einer ganzen Ellipse und eine Region des halben Rechtecks. Anschließend bildet man den Durchschnitt über beide Regionen. Die neue Region stellt aufgrund des erzeugten Durchschnitts eine halbe Ellipse dar, die anschließend mit `PaintRgn` mit dem eingestellten Muster ausgefüllt und deren Rand mit `FrameRgn` ausgezogen wird. Leider ist das Erzeugen von elliptischen Regionen außerordentlich langsam, es kann durchaus Minuten dauern, bis der Sektor gezeichnet ist! Aus diesem Grund wird auch der Mauszeiger durch die Sanduhr ersetzt. Nur verhältnismäßig kleine Regionen sind schnell gezeichnet. Es ist bedauerlich, daß ein solch leistungsfähiges Konzept wie die Regionen in der Praxis daher nur eingeschränkt verwendbar ist.

Die restlichen Funktionen von SHAPE dienen der Initialisierung der Applikation. Sie sind weitgehend Standard und werden nicht mehr weiter erläutert. Oben wurde erwähnt, daß Ressourcen-Daten niemals im Datenspeicher extra gehalten werden sollten. Eine Ausnahme davon wurde am Beginn der Funktion `InitInstance` gemacht: Die Zeichenketten, die für die Ausgabe der Fehlermeldung bei zuwenig Speicher benötigt werden, sollten sich ständig im Speicher befinden. Nur dadurch ist garantiert, daß sie ihren Zweck auch erfüllen. Denn bei zu wenig Speicher ist die Wahrscheinlichkeit groß, daß er auch für das Nachladen der Zeichenketten nicht mehr ausreicht. Das Ergebnis wäre ein leeres Hinweisfeld!

```
/**********************************************************************
 —— H E L P M G R ——— Help-Manager for MS-Windows applications
=======================================================================

This module controls and organizes the help management for a
MS-Windows application which has linked it.

_______________________________________________________________________

Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

Release 1.00 of 89-Jul-29 —— All rights reserved.
**********************************************************************/

#define  NOMINMAX
#include <WINDOWS.H>
#include "HELPMGR.H"

/* further C standard headers */
#include <STDLIB.H>
#include <STRING.H>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)
#define LADDR(r)'(LONG)(LPSTR)(r)

/* local module data values */
static WORD sxChar; /* width of a SYSTEM character */
static WORD syChar; /* height of a standard SYSTEM character */

static HWND hwMain; /* handle of application's main menu */
static HANDLE hiMain; /* handle of application's instance */

/* description line definitions */
#define S_CMD_DESCR 45 /* number of characters in description line */
static HWND hwLine;    /* window handle of command description line */

/* current help mode */
enum {HPM_NONE,HPM_INDEX,HPM_CONTEXT};
static int iHelpMode=HPM_NONE;

static HWND hwDbxHelp; /* window handle for the help dialog boxes */
FARPROC rfdHelp;       /* instance of help dialog box function */
FARPROC rfwSubEdit;    /* subclassing function for "edit" class */
FARPROC rfwOrgEdit;    /* window function of original "edit" class */
static int iTextPos;   /* text position in help text */
static int iTextLen;   /* number of lines in help text */

static int iMenuCmd;    /* last selected menu command */

/* current help text environment */
static int iCurHelpText; /* current set help-text, 0 if no text set */
static int iTextFirst;   /* minimum general help-text index */
static int iTextLast;    /* maximum general help-text index */
static HANDLE hIndex;    /* handle of index table resource */

/* current message/dialog box environment */
static int iCurDlgBox;    /* ident of selected dialog box ident */
static int iCurDlgItem;   /* ident of selected dialog box item */
static int iCurMsgString; /* ident of message string */

/* pointers to message filter hook functions */
FARPROC rfhMsgFilter,rfhPrevMsgFilter;


/**********************************************************************
 H p m S e t C m d L i n e
=======================================================================
This function sets and displayes the command description line after
the application window was resized.

Parameters:
 syClient  is the new vertical size of the client area of
           the application window.
Return:
 none
**********************************************************************/

VOID HpmSetCmdLine(WORD syClient)

 {SetWindowPos
   (hwLine,NULL,-1,syClient-(syChar+4-1),0,0,
    SWP_NOACTIVATE|SWP_NOZORDER|SWP_NOSIZE
   );
  ShowWindow(hwLine,SW_SHOWNA);
 } /* HpmSetCmdLine() */


/**********************************************************************
 G e t T e x t I n d e x
=======================================================================

This function determines the index number of the help text for the
entry <i> in the help keyword index and returns it as a WORD. The
first WORD in the list (at 0) contains the maximum available help text
of the index.

Parameters:
 hwBox  is the handle of the dialog box window.
```

```
Return:
 The index of the text is returned. It is 0 if it was not determined.

**********************************************************************/
static int GetTextIndex(WORD i)

 {WORD FAR *ru;
  int iVal;

  if (hIndex==NULL) return 0; /* memory full => not found */
  /* get index of help text via resource table (type RCDATA) */
  ru=(WORD FAR*)LockResource(hIndex);
  iVal=ru[i]; UnlockResource(hIndex);
  return iVal;
 } /* GetTextIndex() */


/**********************************************************************
 G e t I n d e x P o s
=======================================================================
This function gets the entry in the index list for the text
<iCurHelpText> if this is not 0. The position of this or the previous
position in the index is returned. Extremely the lowest index value is
returned.
Parameters:
 none
Return:
 The position of the current help text is returned. If <iCurHelpText>
 is 0 the first entry in the index is returned.

**********************************************************************/
static int GetIndexPos(VOID)

 {WORD FAR *rmuTable,FAR *ru;
  int iEntry,nEntry;
  WORD iFound=0;

  if (!hIndex||iCurHelpText==0) return 0; /* memory too small */
  /* get index of help text via resource table (type RCDATA) */
  rmuTable=(WORD FAR*)LockResource(hIndex); nEntry=rmuTable[0];
  /* search in index list */
  for (iEntry=0,ru=rmuTable+1;iEntry<nEntry;iEntry++,ru++)
   {if (*ru<=iCurHelpText && *ru>rmuTable[iFound+1])
     {/* set new "previous" element */
      iFound=iEntry;
     } /* if */
   } /* for */
  UnlockResource(hIndex);
  return iFound;
 } /* GetIndexPos() */

/**********************************************************************
 L o a d H e l p I n d e x
=======================================================================
This function reads the help index word list from resource HLS_INDEX
and sets it into the listbox IDLIST of dialog box <hwBox>.
Parameters:
 hwBox  is the handle of the dialog box window.
Return:
 none

**********************************************************************/
static VOID LoadHelpIndex(HWND hwBox)

 {HANDLE hIndex;
  LPSTR rzText,rzEnd;
  BYTE c;

  hIndex=FindResource
   (hiMain,MAKEINTRESOURCE(HLS_INDEX),MAKEINTRESOURCE(HELPLIST));
  if (hIndex) hIndex=LoadResource(hiMain,hIndex);
  if (!hIndex) return; /* memory too small */
  rzText=LockResource(hIndex);
  while (*rzText!=26)
   {/* read next string until end of line */
    rzEnd=rzText+1;
    while ((c=*rzEnd)!=0 && c!='\r' && c!='\n' && c!=26) rzEnd++;
    *rzEnd=0; /* temporary end of string */
    SendDlgItemMessage(hwBox,IDLIST,LB_ADDSTRING,-1,LADDR(rzText));
    while ((c=*rzEnd)<' ' && c!=26) rzEnd++; /* search next string */
    rzText=rzEnd; /* new start of string */
   } /* while */
  UnlockResource(hIndex); /* resource can be moved/discarded */
 } /* LoadHelpIndex() */

/* needed forward references */
BOOL FAR PASCAL fdHelpIndex(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2);
LONG FAR PASCAL fwSubEdit(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2);
BOOL FAR PASCAL fdHelpText(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2);

/**********************************************************************
 S e t I n d e x H e l p
=======================================================================

This function creates the index help modeless dialog box and sets the
new help mode.

Parameters:
 none
Return:
 none
**********************************************************************/
```

◂◂ *Listing 7: Fortsetzung*

Implementierung der Hilfeverwaltung

In diesem und den folgenden Abschnitten soll noch kurz auf die Implementierung der Hilfe-Verwaltung eingegangen werden, wie *Listing 7* sie zeigt. Es sollen nur die für den Windows-Programmierer interessanten Eigenschaften erläutert werden.

Die Funktion `HpmInit` erzeugt alle erforderlichen Daten der Hilfeverwaltung. Das Statuszeilen-Fenster wird als Tochterfenster des Applikationsfenster in der richtigen Größe angelegt. Da es nur Text anzeigen muß, wurde hierfür die vordefinierte Klasse »Static« gewählt. Als Stil wurde ihm ein Rahmen hinzugefügt. Weiterhin legt die Funktion den Systemeingriff zum Umlenken benötigter Systemnachrichten (siehe unten) an.

Eine der wichtigsten Funktionen stellt `HpmProcessHelpMsg` dar, die die für die Hilfeverwaltung interessanten Nachrichten auswertet. Aus der Nachricht `WM_MENUSELECT`, die sonst kaum interessiert, läßt sich die augenblickliche Position im Menü bestimmen, daraus die passende Zeichenkette für die Statuszeilen-Kurzbeschreibung laden und jene über die Windows-Funktion `SetWindowText` in der Statuszeile ausgeben. Als weitere Aufgabe der Funktion dient das Abfangen der Hilfe-Befehle `CMD_HELP_INDEX` und `CMD_HELP_CONTEXT`. Während die erstere den Hilfe-Index einschaltet, ist der letztere Befehl noch nicht realisiert.

Die anderen Funktionen sind zwar teilweise recht komplex, bieten aber insgesamt wenig neues. Jeder, der mit »C« vertraut ist, müßte ihre Funktionsweise nachvollziehen können. Statt dessen gehen wir abschließend noch auf zwei Windows-spezifische Details des Hilfemoduls ein.

Allgemeiner Zugriff auf Ressourcen

Sicherlich interessiert Sie, wie man auf die Ressourcen mit den selbstdefinierten Typen zugreifen kann. Betrachten wir hierzu die Funktion `LoadHelpIndex`, die die Liste des Indexes in ein Listenfeld überträgt. Zunächst muß die Ressource in der Applikationsdatei gefunden werden. Dies erledigt die Windows-Funktion `FindResource`. Als Argument erhält sie den Namen der Ressource und den Typ (hier HELPLIST). Die Funktion liefert einen Bezug zurück, der von Null verschieden ist, falls die Ressource gefunden wurde. Zugegriffen wird auf die Ressource allerdings erst mit der Windows-Funktion `LoadResource`. Sie gibt einen Bezug zurück, der einem Bezug eines globalen Speicherobjekts sehr ähnlich ist. Ähnlich wie bei diesen Objekte, kann man mit der Funktion `LockResource` die Adresse der Ressource erhalten. Erst nach Rückkehr von

dieser Funktion ist die Ressource wirklich in den Speicher geladen worden. `UnlockResource` gibt die Adresse wieder frei, und die Daten können beliebig verschoben oder aus dem Speicher vom System entfernt werden. `LoadResource` überprüft auch, ob sich eine angeforderte Ressource bereits im Speicher befindet und gegebenfalls wird deren Bezug zurückgegeben. Wie man sieht, ist der Zugriff auf allgemeine Ressourcen nicht schwieriger als auf andere angelegte Speicherobjekte. Es lohnt sich also, Daten in Ressourcen anzulegen.

Anzeige der Hilfetexte

Der Index der Hilfeverwaltung wird in einem Listenfeld gehalten, was problemlos ist. Doch wie zeigt man die Hilfetexte an? Eine Art Textanzeigefeld existiert im Windows-System nicht und im »Edit«-Feld könnten die Texte unzulässigerweise geändert werden. Normalerweise müßte man hierfür eine neue Fensterklasse entwickeln, doch dann würde dieser Artikel zur Serie werden. Glücklicherweise gibt es die Unterklassen-Technik *[3]*, mit der sich zwar das Problem nicht perfekt, aber doch ganz brauchbar lösen läßt: Im Dialogfeld `DBX_HELPTEXT` wird aus der »Edit«-Klasse des IDTEXT-Eintrags eine Unterklasse erzeugt, die durch die Funktion `fwSubEdit` realisiert wird. Diese macht nichts anderes, als zu verhindern, daß der Fokus dem Textfenster zugeordnet oder irgendein Zeichen diesem übergeben werden kann. Folglich kann der Text weder markiert noch geändert werden. Zum Blättern im Hilfetext benötigt man allerdings eine Rolleiste. Die implizite Rolleiste der »Edit«-Klasse ist unbrauchbar, denn diese erhält ja keine Tastatureingaben mehr. Als Lösung wurde dem Textfeld eine zusätzliche Rolleiste danebengestellt, die nun das Blättern und Rollen im Textfeld genauso gut wie die Original-Rolleiste durchführt. Nur der Implementierungsaufwand steigt leicht an, da die Kommunikation zwischen beiden Elementen realisiert werden muß.

Systemeingriff in Menü- und Dialogfeldverwaltung

Um bei der Anzeige von Menüs, Dialog- und Hinweisfeldern die Taste F1 abfangen zu können, wird wie in *[2]* ein Systemeingriff (im Original *system hook*) verwendet, den die Funktion `fhMsgFilter` realisiert. Jedesmal, wenn den betreffenden Windows-Elementen eine Nachricht gesandt wird, wird die Funktion aufgerufen und die Nachricht analysiert. Wurde die Taste F1 gedrückt, wird der entsprechende Hilfetext ausgehend vom aktuellen Kontext gesucht und gegebenfalls angezeigt. Anschließend wird die Funktion wieder verlassen und der ursprüngliche

Listing 7: Fortsetzung

```
static VOID SetIndexHelp(VOID)
 {if (!hIndex)
   {/* load index for the first time */
    hIndex=FindResource
     (hiMain,MAKEINTRESOURCE(HLS_INDEX),MAKEINTRESOURCE(RT_RCDATA));
    if (hIndex) hIndex=LoadResource(hiMain,hIndex);
    if (!hIndex) return; /* cannot load */
   } /* if */
  /* create modeless dialog box */
  rfdHelp=MakeProcInstance(fdHelpIndex,hiMain);
  if (CreateDialog
       (hiMain,MAKEINTRESOURCE(DBX_HPMINDEX),hwMain,rfdHelp))
   {/* help box created */
    iHelpMode=HPM_INDEX;
   } /* if */
 } /* SetIndexHelp() */

/*********************************************************************
 S e t C o n t e x t H e l p
======================================================================
This function creates the context help modeless dialog box and sets
the new help mode.
Parameters:
 iNewText  is the index of the new help text.
Return:
 none
*********************************************************************/
static VOID SetContextHelp(int iNewText)

 {iCurHelpText=iNewText; /* set new text */
  rfdHelp=MakeProcInstance(fdHelpText,hiMain); /* dialog function */
  if (CreateDialog
       (hiMain,MAKEINTRESOURCE(DBX_HPMTEXT),hwMain,rfdHelp))
   {/* help box created */
    iHelpMode=HPM_CONTEXT;
   } /* if */
 } /* SetContextHelp() */


/*********************************************************************
 R e s e t H e l p
======================================================================
This function resets the help mode and destroyes the current help
modeless dialog box.
Parameters:
 none
Return:
 none
*********************************************************************/
static VOID ResetHelp(VOID)

 {DestroyWindow(hwDbxHelp); hwDbxHelp=NULL; iHelpMode=HPM_NONE;
  FreeProcInstance(rfdHelp);
 } /* ResetHelp() */


/*********************************************************************
 f d H e l p I n d e x
======================================================================
### Dialog Box function ###
This function processes any messages received by the "Help index"
dialog box.
Parameters:
standard message data
Return:
standard dialog box function value.
*********************************************************************/

BOOL FAR PASCAL fdHelpIndex(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {int i;
  switch (iMsg)
   {case WM_INITDIALOG:
      hwDbxHelp=hw; /* set global variable */
      LoadHelpIndex(hw); /* load resource with help-index keywords */
      /* select current set index entry */
      SendDlgItemMessage(hw,IDLIST,LB_SETCURSEL,GetIndexPos(),0L);
      return TRUE; /* no focus set */
    case WM_COMMAND:
      switch (uP1)
       {case IDLIST:
          if (P2HI!=LBN_DBLCLK) return 0L; /* consumed */
          /* double-click causes IDOK: continue */
        case IDOK:
          /* get selected index */
          i=(int)SendDlgItemMessage(hw,IDLIST,LB_GETCURSEL,0,0L);
          if (i==LB_ERR||(i=GetTextIndex(i+1))==0) i=0; /* no text */
        case IDCANCEL:
          /* close modeless index dialog box */
          ResetHelp();
          if (uP1!=IDCANCEL && i!=0)
           {/* switch into context help */
            iTextFirst=HTX_FIRST;
            iTextLast=GetTextIndex(0); /* set range */
            SetContextHelp(i);
           } /* if */
          return TRUE; /* consumed */
       } /* switch */
      break;
   } /* switch */
  return FALSE;
 } /* fdHelpIndex() */
```

◄◄ *Listing 7: Fortsetzung*

```
/**********************************************************************
 L o a d H e l p T e x t
======================================================================

This function reads the help text from resource with value <iHelpText>
and sets it into the "edit" field IDTEXT of dialog box <hwBox>. The
extra scroll bar for the "edit" field is initialized and set to first
line. The previous and next buttons are disabled if no general help
text is available in the corresponding direction.
Parameters:
 hwBox .... is the handle of the dialog box window.
 iHelpText  is the resource number of the help text.
Return:
 TRUE if the text is loaded, FALSE if not
**********************************************************************/
static BOOL LoadHelpText(HWND hwBox,int iHelpText)

 {HANDLE hIndex;
  LPSTR rzText,rzEnd;
  BYTE c;

  hIndex=FindResource
   (hiMain,MAKEINTRESOURCE(iHelpText),MAKEINTRESOURCE(HELPTEXT));
  if (hIndex) hIndex=LoadResource(hiMain,hIndex);
  if (!hIndex) return FALSE; /* text not loaded */
  rzText=LockResource(hIndex);
  /* first line is title */
  rzEnd=rzText; while ((c=*rzEnd)!=0 && c!='\r' && c!='\n') rzEnd++;
  *rzEnd=0; /* temporary end of title */
  SetDlgItemText(hwBox,IDTITLE,rzText);
  while ((c=*rzEnd)<' ' && c!=26) rzEnd++; /* search start of text */
  /* search end of help text */
  rzText=rzEnd; while ((c=*rzEnd)!=0 && c!=26) rzEnd++;
  *rzEnd=0; /* temporary end of text */
  SetDlgItemText(hwBox,IDTEXT,rzText);
  UnlockResource(hIndex); /* resource can be moved/discarded */
  /* initialize scroll bar */
  iTextLen=(int)SendDlgItemMessage(hwBox,IDTEXT,EM_GETLINECOUNT,0,0L);
  iTextLen=max(iTextLen-10+1,1); iTextPos=1;
  SetScrollRange(GetDlgItem(hwBox,IDTXSRL),SB_CTL,1,iTextLen,FALSE);
  SetScrollPos(GetDlgItem(hwBox,IDTXSRL),SB_CTL,iTextPos,TRUE);
  return TRUE; /* text is loaded */
 } /* LoadHelpText() */


/**********************************************************************
 f w S u b E d i t
======================================================================
### Subclassing window function ###
This function avoids any text change in a window of the "Edit" class.
The text can only be scrolled by using the EM_LINESCROLL message.
Parameters:
standard message data

Return:
standard window function return value.

**********************************************************************/
LONG FAR PASCAL fwSubEdit(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {LONG vRetValue;

  vRetValue=0L;
  switch (iMsg)
   {case WM_SETFOCUS:
    case WM_KEYDOWN:
    case WM_CHAR:
      /* ignore input focus/character input: caret cannot be set */
      return 0L;
   } /* switch */
  /* call original window function */
  return CallWindowProc(rfwOrgEdit,hw,iMsg,uP1,ulP2);
 } /* fwSubEdit() */

/**********************************************************************
 f d H e l p T e x t
======================================================================
### Dialog Box function ###
This function processes any messages received by the DBX_SIZE dialog
box.

Parameters:
standard message data.

Return:
standard dialog box function value.
**********************************************************************/

BOOL FAR PASCAL fdHelpText(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {int i;

  switch (iMsg)
   {case WM_INITDIALOG:
      hwDbxHelp=hw; /* set global variable */
      /* subwindowing of IDTEXT "edit" window: no changes possible */
      rfwOrgEdit=(FARPROC)SetWindowLong
       (GetDlgItem(hw,IDTEXT),GWL_WNDPROC,(LONG)rfwSubEdit);
      /* load resource with specified help text */
      LoadHelpText(hw,iCurHelpText);
      return TRUE; /* no focus set */
```

Datenweg des Windows-Systems fortgesetzt. Der Systemeingriff WH_MSGFILTER ist übrigens einer der wichtigsten und der einzige, der programm- und nicht systemspezifisch ist. Nur er kann in Applikationen verwendet werden, alle anderen müssen in besonderen dynamische Link-Libraries implementiert werden! *[1]*.

Wie bereits oben erwähnt, funktioniert beim Anzeigen von modalen Dialogfeldern nicht die Tastaturbedienung des Hilfetextfelds. Normalerweise müßte sich dies durch einen vorsichtigen Einbau der Funktion `IsDialogMessage` in die Funktion des Systemeingriffs lösen lassen. Leider sperrt sich dagegen aus unerklärlichen Gründen die Windows-Funktion `DialogBox`. Sie weigert sich in Zukunft, bestimmte vorhandene Fenster zu finden. Es wurden unzählige Varianten ausprobiert, aber die Fehlerursache und insbesondere die Vermeidung wurde nicht gefunden. Zu wenig Informationen über die Systemeingriffe und die Semantik der modalen Dialogfelder verbieten eine analytische Lösung. Mich würde es freuen, wenn einer der Leser eine Lösung dazu finden würde.

Bewertung und Ausblick

Das vorgestellte Hilfemodul wurde entwickelt, um einige der Forderungen an die Soforthilfe unter MS-Windows weitgehend applikationsunabhängig zu entwickeln. Es sollte sich dabei nicht um eine umfassende, voll einsatzfähige Lösung handeln, sondern vielmehr um eine Art Prototyp, mit dem untersucht wurde, inwieweit die Möglichkeiten von Windows (Systemeingriffe, Menünachrichten etc.) eine Implementierung ermöglichen. Dies ist meiner Auffassung nach weitgehend gelungen: Windows bietet viele Features, die die Entwicklung eines applikationsunabhängigen Moduls erheblich erleichtern. Die Forderung nach einer Trennung der Organisation der Hilfestruktur und der Hilfetexte einerseits und dem Applikations-Programmcode andererseits konnte vollständig realisiert werden: Die Ressourcen-Implementierung von Windows ist vielseitig, einigermaßen transparent und relativ effizient, da sie auf der Windows-Speicherverwaltung aufbaut. Als etwas problematisch haben sich die geringen Fähigkeiten des Ressourcen-Compilers erwiesen, die zu einer teilweise recht langatmigen Beschreibung der Hilfedaten in der Ressourcen-Datei führten. Überhaupt ist die Erzeugung der Hilfedaten im Augenblick über die Zerlegung der Datei HELP.TXT etwas umständlich. Ich plane daher die Entwicklung eines Generators, welcher aus einer Hilfetextdatei die einzelnen Hilferessourcen automatisch erzeugt und ohne Verwendung des Ressourcen-Compilers die generierten Ressourcen in die EXE-Datei der Applikation einbindet. Diese Textdatei enthält auch alle für die Struktur und die Zuordnung er-

forderlichen Schlüsselwörter, Verweise etc., so daß die Wartung und Erweiterung und insbesondere die Übersetzung in andere Sprachen ziemlich einfach wird. Der Text wird im Rich-Text-Format erstellt *[4]*, so daß auch verschiedene Schriftgrößen, Attribute wie Unterstreichen oder vielleicht sogar die Einbindung von Grafik in die Hilfetexte möglich werden.

Verbesserungsbedürftig ist auch die Präsentation des Hilfemoduls für den Benutzer. So sollten die Texte in kleinerer Proportionalschrift (Helvetica) angezeigt werden, nicht nur weil diese besser lesbar ist, sondern auch, weil auf kleinerem Raum mehr Information untergebracht werden kann. Die Excel-Implementierung von Querverweisen und Glossarbegriffen fehlt ebenfalls noch. Auch ein Inhaltsverzeichnis als Alternative zum Index müßte integriert werden. Ein weiteres Problem stellt der Index dar: Bei größeren Applikationen besitzt er schnell ein paar Hundert Einträge. Auf diese kann man natürlich nicht mehr sinnvoll durch Blättern in einem Listenfeld zugreifen. Man muß vielmehr ein Schnellzugriff vorsehen, für den beispielsweise die ersten drei Buchstaben eines gewünschten Begriffs eingegeben werden. Ferner kann man sich die Implementierung von »Lesezeichen« vorstellen, die der Benutzer sich selbst für Hilfeseiten anlegen kann, in denen er öfters nachschlägt. Als weitere Verbesserung der Hilfeverwaltung würde sich das Einbinden von Hilfetexten durch den Benutzer selbst anbieten, so daß dieser sich seine eigenen Hilfetexten zu seinen persönlichen Problemen mit einem Programm schreiben könnte, so wie er heute in seiner Dokumentation Anmerkungen an den Rand schreibt oder lose Blätter einlegt.

All diese Verbesserungen betreffen jedoch nicht das vorgelegte Konzept der Speicherung der Hilfeinformation in Ressourcen und nur in geringem Umfang die benötigten Fähigkeiten des Windows-Systems. Es sind vielmehr Verbesserungen, die weitgehend im Hilfe-Modul selbst durchgeführt werden müssen und somit auch nicht den Quellcode der zugrundeliegenden Applikation berühren. Die Verbesserungen am Hilfemodul lassen sich mit wenig Aufwand somit mit einem Schlag in alle Applikationen übernehmen. Damit ist noch einmal abschließend gezeigt, wie sinnvoll die Verlagerung der Soforthilfe in ein eigenes Modul ist.

Marcellus Buchheit

[1] Microsoft Windows System Development-Kit, Version 2.

[2] Welch, K.P.: Die Mehrfachdokumentenschnittstelle (MDI), Microsoft System Journal, Juli/August 1989, Seite 5 bis 33.

[3] Buchheit, Marcellus: Einführung in Fensterunterklassen, Microsoft System Journal, März/April 1989, Seite 67 bis 81.

[4] Andrews, Nancy: Rich-Text-Format: Ein Standard erleichtert den Textaustausch. Microsoft System Journal, November 1987, Seite 75 bis 81.

Listing 7: Fortsetzung

```
    case WM_VSCROLL:
      switch (uP1)
       {case SB_TOP:
          i=1; break;
        case SB_BOTTOM:
          i=iTextLen; break;
        case SB_LINEDOWN:
          i=min(iTextPos+1,iTextLen); break;
        case SB_LINEUP:
          i=max(iTextPos-1,1); break;
        case SB_PAGEDOWN:
          i=min(iTextPos+10,iTextLen); break;
        case SB_PAGEUP:
          i=max(iTextPos-10,1); break;
        case SB_THUMBPOSITION:
          i=P2LO; break;
        default:
          /* ignore */
          return FALSE;
       } /* switch */
      /* send scrolling command to edit field */
      if (i!=iTextPos)
       {SendDlgItemMessage
         (hw,IDTEXT,EM_LINESCROLL,0,MAKELONG(i-iTextPos,0));
        iTextPos=i;
        SetScrollPos(GetDlgItem(hw,IDTXSRL),SB_CTL,iTextPos,TRUE);
       } /* if */
      return FALSE;
    case WM_COMMAND:
      switch (uP1)
       {case IDPREV:
          if (iCurHelpText<=iTextFirst)
           {/* at start of table: invalid selection */
            MessageBeep(0);
           }
          else
           {/* load predecessor help-text if possible */
            i=iCurHelpText-1;
            if (LoadHelpText(hw,i)) iCurHelpText=i;
           } /* if */
          break;
        case IDNEXT:
          if (iCurHelpText>=iTextLast)
           {/* at end of table: invalid selection */
            MessageBeep(0);
           }
          else
           {/* load successor help-text if possible */
            i=iCurHelpText+1;
            if (LoadHelpText(hw,i)) iCurHelpText=i;
           } /* if */
          break;
        case IDOK:
        case IDCANCEL:
          /* close modeless index dialog box */
          DestroyWindow(hw); hwDbxHelp=NULL; iHelpMode=HPM_NONE;
          FreeProcInstance(rfdHelp);
          if (uP1==IDOK) SetIndexHelp();
          return TRUE; /* consumed */
       } /* switch */
      break;
   } /* switch */
  return FALSE;
 } /* fdHelpText() */

/*******************************************************************
 S w i t c h I n d e x H e l p
===================================================================
This function activates the index help.
Parameters:
 none
Return:
 none
*******************************************************************/

static VOID SwitchIndexHelp(VOID)

 {switch (iHelpMode)
   {case HPM_NONE:
      iCurHelpText=0; SetIndexHelp(); iHelpMode=HPM_INDEX; break;
    case HPM_INDEX:
      /* activate context help window */
      SetFocus(hwDbxHelp); break;
    case HPM_CONTEXT:
      ResetHelp(); SetIndexHelp(); break;
   } /* switch */
 } /* SwitchIndexHelp() */

/*******************************************************************
 S w i t c h C o n t e x t H e l p
===================================================================
This function activates the context help with the help text
<iHelpText>.
Parameters:
 iNewText .. is the index of the new help text.
 iAreaFirst  is the first value of the current help text area. If this
             is 0, for both bound values a default area is used.
 iAreaLast   is the last value of the current help text area. If the
             value is 0, a default value is used.
Return:
 none
*******************************************************************/
```

```
static VOID SwitchContextHelp
 (int iNewText,int iAreaFirst,int iAreaLast)

 {/* set first and last bound of current help text area */
  if (iAreaFirst==0)
   {/* use default values */
    if (iNewText>=HTX_FIRST && iNewText<HTX_SPECIAL)
     {/* standard help text area (general, menu etc) */
      iTextFirst=HTX_FIRST; iTextLast=GetTextIndex(0);
     }
    else
     {/* no range */
      iTextFirst=iNewText; iTextLast=iNewText;
     } /* if */
   }
  else
   {/* set both values */
    iTextFirst=iAreaFirst; iTextLast=iAreaLast;
   } /* if */
  switch (iHelpMode)
   {case HPM_NONE:
      SetContextHelp(iNewText); break;
    case HPM_INDEX:
      ResetHelp(); SetContextHelp(iNewText); break;
    case HPM_CONTEXT:
      /* change text if needed, activate context help window */
      if (iCurHelpText)
       {/* change help text */
        LoadHelpText(hwDbxHelp,iNewText); iCurHelpText=iNewText;
       } /* if */
      SetFocus(hwDbxHelp); break;
   } /* switch */
 } /* SwitchContextHelp() */



/**********************************************************************
 H p m P r o c e s s H e l p M s g
======================================================================

This function processes all messages which are received by the
application window function if they are of interest for the help
manager.

Parameters:
 hw .. is the window handle of the application window.
 iMsg  is the message which was received by the application window
       function.
 uP1   is the word parameter of the message.
 ulP2  is the long word parameter of the message.

Return:
 If a non-zero value is returned, the function has consumed the
 message. Otherwise the standard function DefWndProc must be called.

**********************************************************************/

LONG HpmProcessHelpMsg(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {BYTE z[S_CMD_DESCR+1];
  WORD uHcs=0;

  switch (iMsg)
   {case WM_MENUSELECT:
      if (P2HI!=NULL)
       {if (P2LO&MF_POPUP)
         {/* popup-window header: get code of first command in menu */
          uHcs=GetMenuItemID(uP1,0);
         }
        else if (P2HI!=NULL)
         {/* menu command selected: determine description string */
          uHcs=uP1;
         } /* if */
        iMenuCmd=uHcs; /* command code of menu */
        if (P2LO&MF_POPUP) iMenuCmd--; /* popup-code */
       } /* if */
      if (uHcs>=0xF000 && P2LO&MF_SYSMENU)
       {/* determine predefined entries in system menu */
        iMenuCmd=uHcs; /* command code of menu */
        switch (uHcs)
         {case SC_RESTORE:
            uHcs=HCS_SYSREST; break;
          case SC_MOVE:
            uHcs=HCS_SYSMOVE; break;
          case SC_SIZE:
            uHcs=HCS_SYSSIZE; break;
          case SC_MINIMIZE:
            uHcs=HCS_SYSMIN; break;
          case SC_MAXIMIZE:
            uHcs=HCS_SYSMAX; break;
          case SC_CLOSE:
            uHcs=HCS_SYSCLOSE; break;
          default:
            uHcs=0; break;
         } /* switch */
       }

      else
       {/* other command: convert to help command string */
        uHcs+=2000;
       } /* if */
      if (uHcs!=0)
       {/* command exists: load description string */
        if (P2LO&MF_POPUP) uHcs--; /* use predecessor for popup */
        LoadString(hiMain,uHcs,z,sizeof(z));
       }
      else
       {/* string is empty */
        z[0]=0;
       } /* if */
      /* send string to command description line */
      SetWindowText(hwLine,z);
      break;
    case WM_COMMAND:
      if (uP1==CMD_HELP_INDEX)
       {/* switch index help (on/off) */
        SwitchIndexHelp();
       }
      else if (uP1==CMD_HELP_CONTEXT)
       {/* activate context help: currently not implemented */
       } /* if */
      break; /* not consumed */
   } /* switch */
  return 0L; /* not consumed */
 } /* HpmProcessHelpMsg() */


/**********************************************************************
 S w i t c h D i a l o g H e l p
======================================================================

This function finds the help text for the current selected dialog box
item. If it is found the help text is displayed and TRUE is returned.
Otherwise FALSE is returned.

Parameters:
 none
Used variables:
 iCurDlgBox,iCurDlgItem
Return:
 none
**********************************************************************/
static int SwitchDialogHelp(VOID)

 {HANDLE hTab;
  WORD FAR *ru;
  int iFirst,iLast,iText;
  BOOL bLastDlg;
  HWND hw;
  hTab=FindResource
   (hiMain,MAKEINTRESOURCE(HLS_DIALOG),MAKEINTRESOURCE(RT_RCDATA));
  if (hTab) hTab=LoadResource(hiMain,hTab);
  if (!hTab) return 0; /* not found or memory too small */
  /* get index of help text via resource table (type RCDATA) */
  ru=(WORD FAR*)LockResource(hTab);
  hw=GetFocus(); /* assumed that selected item in dialog box */
  /* window must be child window otherwise no dialog item */
  if (!(GetWindowLong(hw,GWL_STYLE)&WS_CHILD)) return FALSE;
  /* read ident of child window */
  iCurDlgItem=GetWindowWord(hw,GWW_ID);
  /* search item in table */
  bLastDlg=FALSE; iText=0;
  do
   {if (*ru!=iCurDlgBox && *ru)
     {/* skip dialog box area */
      ru++; /* skip dialog box ident */
      while (*ru) ru+=2; /* skip items of dialog box */
      ru++; /* skip end of dialog box area */
     }
    else
     {/* dialog box matched: find item */
      if (!*ru) bLastDlg=TRUE; /* default box => last chance */
      ru++; /* skip dialog box ident */
      iFirst=*(ru+1); /* first element in dialog box text area */
      /* check if dialog item matches */
      while (*ru)
       {/* search item */
        if (*ru==iCurDlgItem)
         {/* item found => get ident of help text, exit loop */
          iText=*(ru+1);
          /* search last element */
          while (*ru) ru+=2; /* skip items of dialog box */
          iLast=*(ru-1); /* get last element of dialog box text area */
          goto Found; /* entry is found */
         } /* if */
        ru+=2; /* next item */
       } /* while */
      ru++; /* skip end of dialog box area */
     } /* if */
   }
  while (!bLastDlg);
  Found:
  UnlockResource(hTab); FreeResource(hTab);
  if (iText!=0)
   {/* help text found: display it */
    if (bLastDlg||iText<HTX_SPECIAL)
     {/* no range possible or general text */
      iFirst=0;
     } /* if */
    SwitchContextHelp(iText,iFirst,iLast); return TRUE; /* found */
   } /* if */
  return FALSE;
 } /* SwitchDialogHelp() */
```

◄ *Listing 7: Fortsetzung*

▶ *Listing 7: Fortsetzung*

```
/***********************************************************************
 S w i t c h M e n u H e l p
========================================================================

This function finds the help text for the current selected menu. If it
is found the help text is displayed and TRUE is returned. Otherwise
FALSE is returned.

Parameters:
 none
Return:
 none
***********************************************************************/

static int SwitchMenuHelp(VOID)
 {HANDLE hTab;
  WORD FAR *ru;
  hTab=FindResource
   (hiMain,MAKEINTRESOURCE(HLS_MENU),MAKEINTRESOURCE(RT_RCDATA));
  if (hTab) hTab=LoadResource(hiMain,hTab);
  if (!hTab) return 0; /* not found or memory too small */
  /* get index of help text via resource table (type RCDATA) */
  ru=(WORD FAR*)LockResource(hTab);
  while (*ru)
   {if (*ru==iMenuCmd) break; /* command found */
    ru+=2; /* next pair */
   } /* while */
  UnlockResource(hTab); FreeResource(hTab);
  if (*ru==iMenuCmd)
   {/* help text found: display it */
    SwitchContextHelp(*(ru+1),0,0); return TRUE; /* found */
   } /* if */
  return FALSE;
 } /* SwitchMenuHelp() */

/***********************************************************************
 S w i t c h M e s s a g e H e l p
========================================================================
This function finds the help text for the current displayed message
box. If it is found the help text is displayed and TRUE is returned.
Otherwise FALSE is returned.
Parameters:
 none
Used variables:
 iCurDlgBox,iCurDlgItem
Return:
 none
***********************************************************************/
static int SwitchMessageHelp(VOID)
{HANDLE hTab;
 WORD FAR *ru;
 int iText;
 BOOL bLastDlg;
 if (!iCurMsgString) return FALSE; /* no string */
 hTab=FindResource
  (hiMain,MAKEINTRESOURCE(HLS_MESSAGE),MAKEINTRESOURCE(RT_RCDATA));
 if (hTab) hTab=LoadResource(hiMain,hTab);
 if (!hTab) return 0; /* not found or memory too small */
 /* get index of help text via resource table (type RCDATA) */
 ru=(WORD FAR*)LockResource(hTab); iText=0;
 while (*ru)
  {/* outer loop: search selected message string */
   if (*ru++!=iCurMsgString)
    {/* skip message string area */
     while (!*ru)
      {/* skip dialog box area */
       ru++; /* skip dialog box ident */
       while (*ru) ru+=2; /* skip items of dialog box */
       ru++; /* skip end of dialog box area */
      } /* while */
     ru++; /* skip end of message string area */
    }
   else
    {/* message string matched: find dialog box & item */
     bLastDlg=FALSE;
     do
      {if (*ru!=iCurDlgBox && *ru)
        {/* skip dialog box area */
         ru++; /* skip dialog box ident */
         while (*ru) ru+=2; /* skip items of dialog box */
         ru++; /* skip end of dialog box area */
        }
       else
        {/* dialog box matched: find item */
         if (!*ru) bLastDlg=TRUE; /* default box => last chance */
         ru++; /* skip dialog box ident */
         /* check if dialog item matches */
         while (*ru)
          {/* search item */
           if (*ru==iCurDlgItem)
            {/* item found => get ident of help text, exit loop */
             iText=*(ru+1); goto Found;
            } /* if */
           ru+=2; /* next item */
          } /* while */
         ru++; /* skip end of dialog area */
        } /* if */
      }
```

```
      while (!bLastDlg);
     } /* if */
   } /* while */
  Found:
  UnlockResource(hTab); FreeResource(hTab);
  if (iText!=0)
   {/* help text found: display it */
    SwitchContextHelp(iText,0,0); return TRUE; /* found */
   } /* if */
  return FALSE;
 } /* SwitchMessageHelp() */

/*********************************************************************
 f h M s g F i l t e r
======================================================================

### Hook filter procedure ###

This function is the filter function for all dialog-box/message-box/
menu messages to check if a help-manager button is pressed.

If the WM_KEYDOWN message is received and the key VK_F1 is pressed,
the context help is displayed or actualized.
All others messages are not consumed.
*********************************************************************/

DWORD FAR PASCAL fhMsgFilter(int iCode,WORD uP,MSG FAR *rwm)
 {int iMsg;

  if (iCode>=0)
   {iMsg=rwm->message; /* current message */
    if (iMsg==WM_KEYDOWN && rwm->wParam==VK_F1 && rwm->hwnd!=hwDbxHelp)
     {/* F1-button pressed: activate/change help */
      switch (iCode)
       {case MSGF_DIALOGBOX:
          /* determine help text/dialog box entry and display it */
          if (SwitchDialogHelp())
           {/* message consumed */
            rwm->message=WM_NULL; return 0L;
           } /* if */
          break; /* not consumed */
        case MSGF_MENU:
          /* determine help text/current menu entry and display it */
          if (SwitchMenuHelp())
           {/* message consumed: close menu by replacing F1 by ESC */
            rwm->message=WM_KEYDOWN; rwm->wParam=VK_ESCAPE; return 0L;
           } /* if */
          break; /* not consumed */
        case MSGF_MESSAGEBOX:
          /* determine help text/current message box and display it */
          if (SwitchMessageHelp())
           {/* message consumed */
            rwm->message=WM_NULL; return 0L;
           } /* if */
          break; /* not consumed */
       } /* switch */
     } /* if */
   } /* if */
  /* not processed => use default */
  return DefHookProc
          (iCode,uP,LADDR(rwm),(FARPROC FAR*)&rfhPrevMsgFilter);
 } /* fhMsgFilter() */

/*********************************************************************
 H p m C h e c k M e s s a g e
======================================================================

This function checks if the message [*rwm] is for the help manager.
If so the message is processed and TRUE is returned.
Otherwise FALSE is returned.

Parameters:
 rwm  is the address of the window message.

Return:
 TRUE if the message was processed by the help manager
 and FALSE if not.

*********************************************************************/

BOOL HpmCheckMessage(LPMSG rwm)

 {BOOL b=FALSE;
  if (hwDbxHelp)
   {/* check if help dialog box message */
    b=IsDialogMessage(hwDbxHelp,rwm);
   } /* if */
  return b;
 } /* HpmCheckMessage() */

/*********************************************************************
 H p m S e t M s g B o x E n v
======================================================================
This function sets the environment constants for the message box which
is displayed in near future. The values are used if the user presses the
[SHIFT][F1] keyboard combination to get help about the message.
```

◄ *Listing 7: Fortsetzung*

► *Listing 7: Ende*

►► *Listing 8: Die Datei HELP.TXT enthält alle Hilfetexte, die von SHAPE verwendet werden.*

```
Parameters:
 iItem    is the ident of the item in the dialog box or 0 if no
          special item is selected.
 iString  is the ident of the string which is displayed in the
          message box. If this value is 0, no help can be chosen about
          the displayed message (for example if memory to small).

Return:
 none

*********************************************************************/


VOID HpmSetMsgBoxEnv(WORD iItem,WORD iString)

 {/* set values to module variables */
  iCurDlgItem=iItem; iCurMsgString=iString;
 } /* HpmSetMsgBoxEnv() */



/*********************************************************************
 H p m S e t D l g B o x E n v
 =====================================================================

This function sets the identification value of the current used dialog
box.
This is the value of the resource file description of the dialog box.
The value is used if the user presses the [SHIFT][F1] keyboard
combination to get help about the current dialog item of a message
string.

Parameters:
 iDlgBox  is the ident of the dialog box or 0 if no special
          dialog box is selected.

Return:
 none

*********************************************************************/


VOID HpmSetDlgBoxEnv(WORD iDlgBox)

 {/* set value to module variable */
  iCurDlgBox=iDlgBox;
 } /* HpmSetDlgBoxEnv() */



/*********************************************************************
 H p m I n i t
 =====================================================================

This function initializes the help manager. If creates a small child
window at the left bottom corner of the client area of the window
<hwAppl>.

Parameters:
 hwAppl  is the window handle of the application in which client area
         the help manager's command description line is displayed.
 hiAppl  is the module instance handle of the application.

Return:
 TRUE if help manager is initialized FALSE if not.

*********************************************************************/


BOOL HpmInit(HWND hwAppl,HWND hiAppl)

 {TEXTMETRIC wtm;
  HDC hdc;

  hwMain=hwAppl; /* store window handle of application's menu */
  hiMain=hiAppl; /* set instance of application */
  /* determine size of standard display characters */
  hdc=CreateIC("Display",NULL,NULL,NULL);
  GetTextMetrics(hdc,&wtm);
  sxChar=wtm.tmAveCharWidth; syChar=wtm.tmHeight;
  DeleteDC(hdc);
  /* create command descriptor line window */
  hwLine=CreateWindow
   ("static","",WS_BORDER|SS_LEFT|WS_CLIPSIBLINGS|WS_CHILD,
    0,0,S_CMD_DESCR*sxChar,syChar+4,hwAppl,NULL,hiAppl,NULL
   );
  if (!hwLine) return FALSE; /* not created => error */
  /* set subclass for text browsing */
  rfwSubEdit=MakeProcInstance((FARPROC)fwSubEdit,hiMain);
  /* set hook function for dialog-box/menu/message-box analysing */
  rfhMsgFilter=MakeProcInstance((FARPROC)fhMsgFilter,hiMain);
  rfhPrevMsgFilter=SetWindowsHook(WH_MSGFILTER,rfhMsgFilter);
  return TRUE;
 } /* HpmInit() */



/* ============================
   End of WINDOWS module HELPMGR
   ============================
*/
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%INDEX.HTX
Applikation
Farbe
Fⁿllmuster
Gr÷▀e
Hilfe
L÷schen
Objekte
Optionen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%APPL.HTX
Applikation
Die Applikation SHAPE dient zum Anzeigen festgeleg-
ter geometrischer Elemente. Es k÷nnen verschiedene
Darstellungseigenschaften fⁿr die Elemente gewΣhlt
werden, so die Gr÷▀e und Farbe und das Muster, mit
der sie gefⁿllt werden.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%OBJECT.HTX
Objekte
Mit SHAPE k÷nnen folgende Objekte gezeichnet wer-
den: Ellipse, Raute und (Ellipsen-)Sektor. Es wird
jeweils nur eines der gezeichneten Objekte ange-
zeigt, dieses allerdings in einstellbarer Gr÷▀e
und Farbe und das Fⁿllmuster kann ausgewΣhlt wer-
den. Die Auswahl des Objekts geschieht im Menⁿ
╗Zeichnen½, die anderen Einstellungen erfolgen im
Menⁿ ╗Optionen½.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%MNUDRAW.HTX
Menⁿ ╗Zeichnen½
Mit Hilfe dieses Menⁿs k÷nnen folgende Objekte ge-
zeichnet werden: Ellipse, Raute und (Ellipsen-)
Sektor. Hierbei wird das Objekt in der eingestell-
ten Gr÷▀e und Farbe gezeichnet und das ausgesuchte
Fⁿllmuster verwendet.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDELIP.HTX
Befehl ╗Ellipse½
Mit diesem Befehl wird eine Ellipse in der ein-
gestellten Farbe gezeichnet und mit dem eingege-
benen Fⁿllmuster ausgefⁿllt. Die Breite und H÷he
der Ellipse ergibt sich aus den eingestellten
horizontalen und vertikalen Gr÷▀enwerten.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDRHOMB.HTX
Befehl ╗Raute½
Mit diesem Befehl wird eine Raute in der ange-
gebenen Farbe gezeichnet und mit dem angegebenen
Fⁿllmuster ausgefⁿllt. Die Breite und H÷he der
Raute ergibt sich aus den angegebenen hori-
zontalen und vertikalen Gr÷▀enwerten.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDSECT.HTX
Befehl ╗Sektor½
Mit diesem Befehl wird ein halber Sektor einer
Ellipse in der angegebenen Farbe gezeichnet
und mit dem angegebenen Fⁿllmuster ausgefⁿllt.
Die Breite und H÷he des Quadrats ergibt sich
aus dem angegebenen horizontalen Gr÷▀enwert.
Das Zeichnen gr÷▀erer Sektoren kann mehrere
Minuten dauern!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDEND.HTX
Befehl ╗Ende½
Mit diesem Befehl wird die Applikation SHAPE
beendet.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%OPTION.HTX
Optionen
Mit SHAPE k÷nnen die Objekte in verschiedenem
Erscheinungsbild gezeichnet werden. Es kann die
Gr÷▀e des Objektes, die Farbe und das Fⁿllmuster
in beschrΣnktem Umfang eingestellt werden. Dies
geschieht mit Befehlen im Menⁿ ╗Optionen½.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%MNUOPT.HTX
Menⁿ ╗Optionen½
Mit Befehlen in diesem Menⁿ kann das Er-
scheinungsbild des zu zeichnenden Objekts
in beschrΣnktem Umfang eingestellt werden.
Hierbei kann die Gr÷▀e des Objektes, die
Farbe und das Fⁿllmuster in beschrΣnktem
Umfang eingestellt werden.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%SIZE.HTX
Gr÷▀e
Mit SHAPE k÷nnen die Objekte in unterschiedlicher
Gr÷▀e gezeichnet werden. Hierzu ist im Menⁿ
╗Optionen½ der Befehl ╗Gr÷▀e½ zu wΣhlen.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDSIZE.HTX
Befehl ╗Gr÷▀e½
Dieser Befehl ruft ein Dialogfeld aus, mit dem
die Objektgr÷▀e fⁿr die weiteren Zeichenopera-
tionen ausgewΣhlt werden kann. Die Gr÷▀e wird
horizontal und vertikal als Prozentwert der
aktuellen Fenstergr÷▀e von SHAPE angegeben.
Wird die Fenstergr÷▀e verΣndert, Σndert sich
entsprechend auch die Gr÷▀e des gezeichneten
Objekts.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%COLOR.HTX
Farbe
Mit SHAPE k÷nnen die Objekte in unterschiedlicher
Farbe gezeichnet werden. Hierzu ist im Menⁿ
╗Optionen½ der Befehl ╗Farbe½ zu wΣhlen.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDCOLOR.HTX
Befehl ╗Farbe½
Dieser Befehl ruft ein Dialogfeld auf, mit dem
der Rot- Grⁿn- und Blau-Anteil der Objektfarbe
eingestellt werden kann. Der eingestellte Wert
wird fⁿr die weiteren Zeichenoperationen ver-
wendet.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FILL.HTX
Fⁿllmuster
Mit SHAPE k÷nnen die Objekte mit unterschiedlichen
Mustern gefⁿllt werden. Hierzu ist im Menⁿ
╗Optionen½ der Befehl ╗Fⁿllmuster½ zu wΣhlen.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDFILL.HTX
Befehl ╗Fⁿllmuster½
Dieser Befehl ruft ein Dialogfeld aus, mit dem
das Fⁿllmuster fⁿr die weiteren Zeichenopera-
tionen ausgewΣhlt werden kann. Das Fⁿllmuster
wird in der eingestellten Farbe erzeugt. Es
kann zwischen leerem, diagonal gestreiften
und vollem Fⁿllmuster gewΣhlt werden.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CLEAR.HTX
L÷schen
Ein gezeichnetes Element kann mit dem Befehl
╗L÷schen½ in der Menⁿleiste gel÷scht werden.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDCLR.HTX
Menⁿ/Befehl ╗L÷schen½
Hiermit wird der Fensterinhalt gel÷scht. Ein vor-
her gezeichnetes Objekt wird entfernt.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%HELP.HTX
Hilfe
Das Programm ist mit einer leistungsfΣhigen Hilfe-
funktion ausgestattet. Mit dem Befehl ╗Hilfe-Index½
im Menⁿ ╗Hilfe½ wird der Index angezeigt, der
Stichw÷rter fⁿr bestimmte Probleml÷sungen besitzt.
Die Stichw÷rter verweisen in eine Sammlung von
Hilfetexten, die sequentiell miteinander verbunden
sind. Der Hilfe-Index kann auch mit der Taste [F1]
angewΣhlt werden.
Mit [F1] kann innerhalb von Menⁿs, Dialog- und
Hinweisfeldern Hilfe angefordert werden. Der ange-
zeigte Hilfetext bezieht sich auf das Element,
welches augenblicklich selektiert ist.
Der Befehl ╗Kontexthilfe½ dient zur ErlΣuterung von
Dialogobjekten im Menⁿ oder im Applikationsfenster
(auch im Rahmen). Er ist z.Zt. nicht implementiert.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%MNUHELP.HTX
Menu Hilfe
Dieses Menⁿ wΣhlt die Soforthilfe aus. Es kann
zwischen dem Hilfe-Index mit Schlⁿsselw÷rtern und
der Kontext-Hilfe gewΣhlt werden.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDHPIX.HTX
Befehl ╗Hilfe-Index½
Dieser Befehl zeigt eine Liste von Schlⁿsselw÷r-
tern, die Anwendungsprobleme oder allgemeine Fakten
betreffen. Nach der Auswahl eines der Schlⁿsselw÷r-
ter wird der entsprechende Hilfetext angezeigt. Von
diesem Text aus kann durch Vor- und ZurⁿckblΣttern
verwandte Informationen im Hilfeverzeichnis abge-
rufen werden.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CMDHPCTX.HTX
Befehl ╗Kontexthilfe½
Der Befehl ist zur Zeit nicht implementiert. SpΣter
verwandelt er den Mauszeiger in einen Zeiger mit
Fragezeichen, mit dem die Bedeutung von Feldern
innerhalb des Applikationsfensters, des Menⁿs oder
auch des Fensterrahmens erfragt werden k÷nnen.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%VERT.HTX
Textfeld ╗Vertikal½
Mit diesem Feld wird die vertikale Gr÷▀e des zu
zeichnenden Objekts bestimmt. Sie wird als Prozent-
wert der vertikalen Fensterinhalts angegeben.
ZulΣssig sind ganzzahlige Werte zwischen 0 und 100.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%HORZ.HTX
Textfeld ╗Horizontal½
Mit diesem Feld wird die horizontale Gr÷▀e des zu
zeichnenden Objekts bestimmt. Sie wird als Prozent-
wert der vertikalen Fensterinhalts angegeben.
ZulΣssig sind ganzzahlige Werte zwischen 0 und 100.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%SIZEOK.HTX
SchaltflΣche ╗OK½
Hiermit wird die Eingabe des Dialogfelds abgeschlos-
sen und die eingestellten Gr÷▀enwerte fⁿr das nach-
folgende Zeichnen von Objekten abgespeichert.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%RED.HTX
Rolleiste ╗Rot½
Hiermit wird der Rotanteil fⁿr das Objektzeichnen
eingestellt. Er kann zwischen 0 fⁿr schwarz ⁿber 50
fⁿr halben Grauanteil bis zu 100 fⁿr sattes Rot
eingestellt werden. Die aktuelle Prozentwert wird
rechts von der Rolleiste angezeigt.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%GREEN.HTX
Rolleiste ╗Grⁿn½
Hiermit wird der Grⁿnanteil fⁿr das Objektzeichnen
eingestellt. Er kann zwischen 0 fⁿr schwarz ⁿber 50
fⁿr halben Grauanteil bis zu 100 fⁿr sattes Grⁿn
eingestellt werden. Die aktuelle Prozentwert wird
rechts von der Rolleiste angezeigt.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%BLUE.HTX
Rolleiste ╗Blau½
Hiermit wird der Blauanteil fⁿr das Objektzeichnen
eingestellt. Er kann zwischen 0 fⁿr schwarz ⁿber 50
fⁿr halben Grauanteil bis zu 100 fⁿr sattes Blau
eingestellt werden. Die aktuelle Prozentwert wird
rechts von der Rolleiste angezeigt.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%COLOROK.HTX
SchaltflΣche ╗OK½
Hiermit wird die Eingabe des Dialogfelds abge-
schlossen und die eingestellen Farbenwerte fⁿr das
nachfolgende Zeichnen von Objekten abgespeichert.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FILL1.HTX
Rundes Optionsfeld ╗Muster 1½
Hiermit wird angegeben, da▀ das zu zeichnende Ob-
jekt nicht gefⁿllt werden soll sondern der Fenster-
hintergrund im Innern des Objekts erscheint.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FILL2.HTX
Rundes Optionsfeld ╗Muster 2½
Hiermit wird angegeben, da▀ das zu zeichnende Ob-
jekt mit einem schraffierten Muster gefⁿllt werden
soll. Das Muster besitzt die rechts angegebene
Farbe. Ist das Muster nicht sichtbar, ist die
Hintergrundfarbe als Objektfarbe gewΣhlt worden und
das Objekt somit beim Zeichnen nicht sichtbar.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FILL3.HTX
Rundes Optionsfeld ╗Muster 3½
Hiermit wird angegeben, da▀ das zu zeichnende
Objekt mit einem gleichmΣ▀igen Farbton gefⁿllt
werden soll.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FILLOK.HTX
SchaltflΣche ╗OK½
Hiermit wird die Eingabe des Dialogfelds abge-
schlossen und das eingestelle Fⁿllmuster fⁿr das
nachfolgende Zeichnen von Objekten abgespeichert.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CANCEL.HTX
SchaltflΣche ╗Abbruch½
Hiermit wird die Eingabe des Dialogfelds abge-
schlossen, die eingestellten Werte aber verworfen
und nicht ⁿbernommen.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%HORZVAL.HTX
Illegaler numerischer Wert
Der Horizontal-Wert mu▀ ein ganzzahliger numeri-
scher Wert im Bereich von 0 bis 100 sein. Andere
Werte oder nichtnumerische Zeichen sind nicht
zulΣssig.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%VERTVAL.HTX
Illegaler numerischer Wert
Der Vertikal-Wert mu▀ ein ganzzahliger numerischer
Wert im Bereich von 0 bis 100 sein. Andere Werte
oder nichtnumerische Zeichen sind nicht zulΣssig.
```

◄◄ *Listing 8: Fortsetzung*

```
/*********************************************************************
 ———————— S P L I T ———— DOS-Application ————————
*********************************************************************
This program parses the file HELP.TXT into its single help text files.
Copyright 1989 by Marcellus Buchheit, Buchheit software research
All rights reserved.
*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
/* ============
   main function
   ============*/
void main(VOID)
 {FILE *rSrcFile;
  FILE *rDestFile;
  char zLine[80+1];
  char rzFileName[80];
  char *rz;
  int nFiles=0;
  /* open source file */
  rSrcFile=fopen("HELP.TXT","rt");
  if (!rSrcFile)
   {/* file not opened: error message, abort */
    printf("SPLIT: Cannot open file HELP.TXT.\n");
    exit(2); /* unable to start operation */
   } /* if */
  fgets(zLine,sizeof(zLine)-1,rSrcFile);
  while (!feof(rSrcFile))
   {/* skip '%' in header line, create dest. file, read help text */
    rz=zLine; /* pointer to start of line */
    while (*rz=='%') rz++;
    if (rz[strlen(rz)-1]=='\n') rz[strlen(rz)-1]=0;
    strcpy(rzFileName,rz);
    rDestFile=fopen(rzFileName,"w+t");
    if (!rDestFile)
     {/* file not created: error message, abort */
      printf("SPLIT: Cannot create file %s.\n",rzFileName);
      exit(2); /* unable to start operation */
     } /* if */
    fgets(zLine,sizeof(zLine)-1,rSrcFile);
    while (!feof(rSrcFile) && zLine[0]!='%')
     {/* copy text to destination file */
      fputs(zLine,rDestFile);
      fgets(zLine,sizeof(zLine)-1,rSrcFile);
     } /* while */
    /* close file */
    fputc(26,rDestFile);
    if (fclose(rDestFile))
     {/* cannot close destination file */
      printf("SPLIT: Cannot close file %s.\n",rzFileName);
      exit(2); /* unable to start operation */
     } /* if */
    nFiles++;
   } /* while */
  printf("%d help text files created.\n",nFiles);
  exit(0);
 } /* main() */
```

◄ *Listing 9: Das DOS-Programm SPLIT zerteilt HELP.TXT in einzelne Dateien, die vom Ressourcen-Compiler eingelesen werden.*

Bei PC-Software setzen wir weltweit den Standard. Microsoft-Produkte stehen für Leistung und Benutzerfreundlichkeit. Einige Beispiele: die Betriebssysteme MS-DOS und MS OS/2, die Textverarbeitung MICROSOFT WORD oder die WINDOWS-Applikation MICROSOFT EXCEL.
Für unsere **Hauptverwaltung in München** suchen wir

SOFTWARE-TRAINER/IN

Im Rahmen unseres MICROSOFT-Instituts sind Sie verantwortlich für Schulungen über unsere System-Software für Großkunden und renommierte Hardware-Hersteller. Dies umfaßt ein Spektrum von der Vorbereitung der Seminarunterlagen über die Präsentation bis hin zur persönlichen Beratung.
Nach einem DV-orientierten Hochschulstudium haben Sie Ihr pädagogisches Geschick bereits in der Praxis unter Beweis gestellt. Sie haben gute Kenntnisse in einer höheren Programmiersprache, idealerweise in „C".

Wir sind ein modernes und innovatives Unternehmen. Als deutsche Tochter des Weltmarktführers für hochwertige System- und Applikationssoftware im PC-Bereich verzeichnen wir jährlich überdurchschnittliche Zuwachsraten. Unser Arbeitsstil ist kooperativ und leistungsorientiert. Kreativität und Einsatzfreude eröffnen unseren Mitarbeitern vielseitige Entwicklungsmöglichkeiten und sichern den gemeinsamen Erfolg.

SOFTWARE-SPEZIALIST/IN

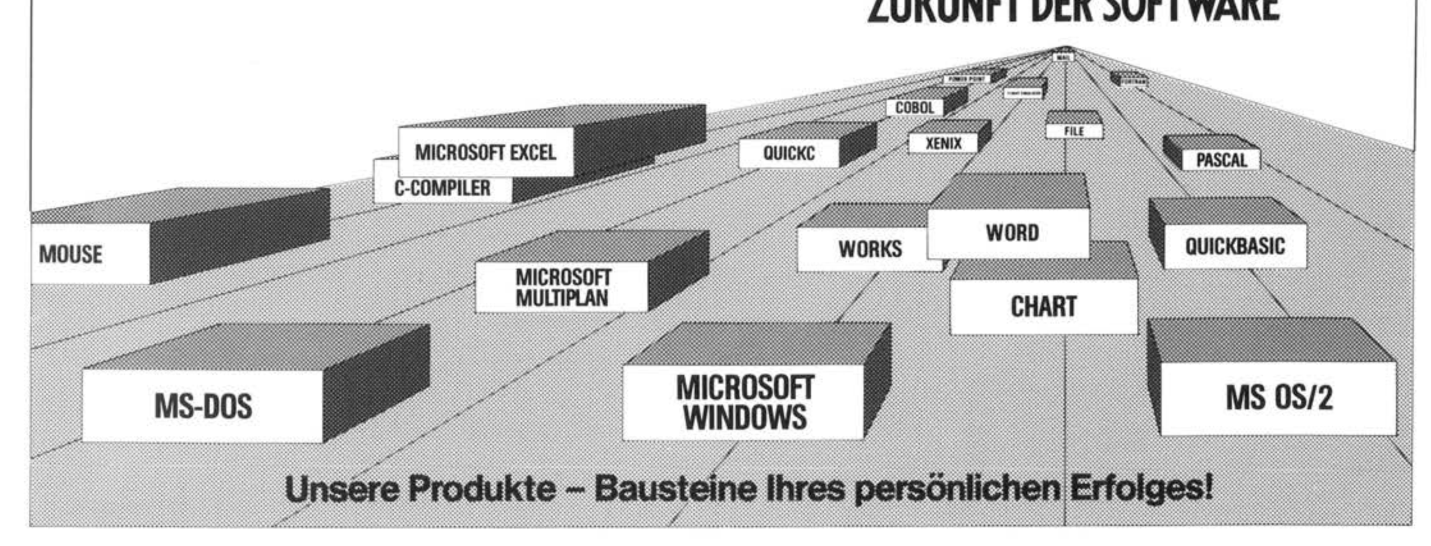
FÜR MICROSOFT-WINDOWS UND MICROSOFT PRESENTATION MANAGER

Sie beraten und unterstützen Software-Entwickler, die unter Microsoft-Windows oder Presentation Manager arbeiten. Dies umfaßt neben der direkten Beratung auch Projektarbeiten bei Software-Entwicklern vor Ort. Außerdem organisieren Sie „Entwicklertreffs" und wirken bei Messen mit.
Nach Abschluß Ihres Informatikstudiums haben Sie in den letzten 3 bis 4 Jahren Berufserfahrung im Bereich Software-Entwicklung für graphische Systeme gesammelt. Sie verfügen über profunde Kenntnisse in „C" und idealerweise auch in Assembler.

Für **beide** Positionen sind sowohl Anwendererfahrungen mit MS-DOS als auch gute Englischkenntnisse Voraussetzung. Außerdem arbeiten Sie gern selbständig und trotzdem teamorientiert.
Wenn Sie in einer dieser Positionen die konsequente Fortsetzung Ihrer Karriere sehen, schicken Sie bitte Ihre aussagefähigen Bewerbungsunterlagen mit Angabe der Gehaltsvorstellung an:

Microsoft GmbH, Personalabteilung,
Edisonstraße 1
8044 Unterschleißheim

Microsoft®
ZUKUNFT DER SOFTWARE



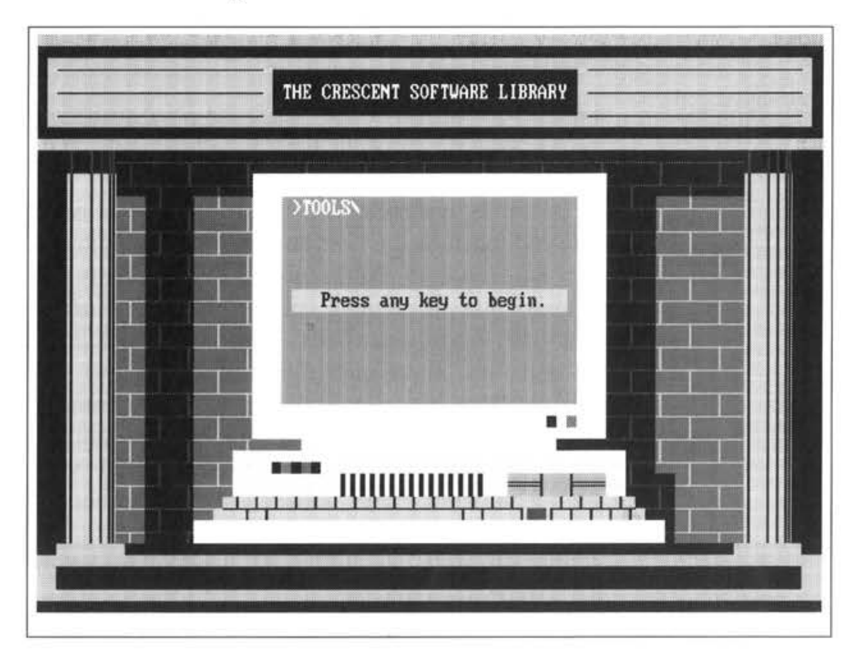
Ein Interview mit Ethan Winer:

Basic als professionelle Programmiersprache

Für die meisten PC-Programmierer sind C und der Makro-Assembler (MASM) die Sprachen der Wahl. Gelegentlich hört man von einem Programmierer, der Basic verwendet. Im allgemeinen jedoch denken professionelle Programmierer, daß Basic eine Spielzeugsprache ist – nett zum Lernen, aber kaum etwas für ernsthafte Programmentwicklung. Diese Einschätzung beginnt sich zu ändern.

Tatsächlich ist Basic im letzten Jahr deutlich erwachsener geworden. Die Benutzeroberfläche von Microsoft QuickBasic bietet eine leistungsfähige Arbeitsumgebung für Anfänger und Fortgeschrittene, und mit dem Microsoft Basic-Compiler Version 6 hat die Sprache inzwischen alle professionellen Features, die ein Programmierer braucht.

Einer von denen, die glauben, daß Basic imstande ist, die Ansprüche professioneller Programmierer zu erfüllen, ist Ethan Winer, Gründer von Crescent Software – einer auf Software-Entwicklung mit Basic spezialisierten Firma – und Entwickler von QuickPak Professional, einer Funktionsbibliothek speziell für Basic. Daneben hat Winer zahlreiche Basic-Artikel in vielen Fachzeitschriften geschrieben.



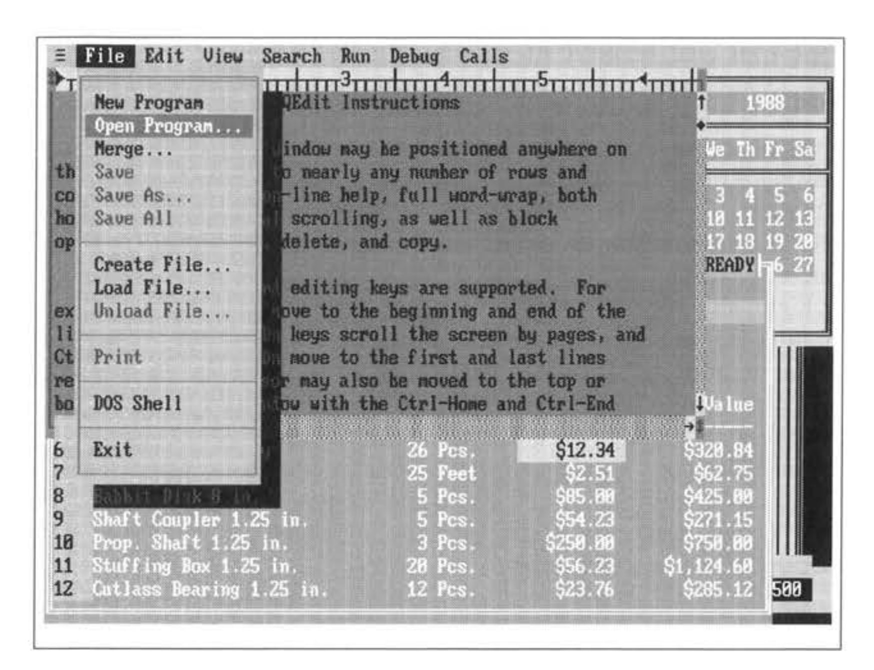
Mit Basic stehen dem Programmierer umfangreiche Möglichkeiten zur Verfügung.

Vor kurzem sprachen wir mit Winer und er erzählte uns, warum er sich für Basic entschieden hat, warum er QuickPak Professional entwickelte, und er gab uns dabei einige Einblicke in technische Aspekte der Basic-Programmierung.

Warum haben Sie und Crescent Software sich für Basic als Programmiersprache entschieden?

Anders als C oder Pascal enthielt Microsoft Basic immer eine Fülle eingebauter Kommandos und Funktionen. Man kann damit anspruchsvolle Grafiken auf einer Vielzahl von Monitoren erzeugen, Dateien und Geräte öffnen und bearbeiten, Musik machen und den Speicher sowie die Hardware-Ports direkt ansprechen. Die Version 4 von Microsoft QuickBasic und der Basic 6.0-Compiler haben weitere Befehle und Funktionen hinzugefügt.

Basic war immer die am einfachsten einzusetzende von allen höheren Programmiersprachen, und die jüngste Version ist ebenso mächtig und leistungsfähig wie C oder Pascal. Darüber hinaus hat Microsoft klar gesagt, daß es beide Compiler weiterentwickeln wird.



Auch Pull-Down-Menüs und mehrfache Fenster sind mit Basic möglich. Man beachte die Scroll-Balken an der Seite des Editorfensters.

Welche der Features, die Basic zu einer professionellen Sprache machen, finden sie besonders nützlich?

Der entscheidende Vorteil von Basic ist, daß es äußerst einfach zu lernen und einzusetzen ist; die Befehle und Funktionen haben sprechende Namen und ihr Zweck ist normalerweise offensichtlich. Anders als bei C oder Assembler ist es so gut wie unmöglich, versehentlich das Betriebssystem oder den Speicher zu überschreiben, ohne daß dadurch die Kontrolle des Programms über die Systemressourcen eingeschränkt würden. PEEK und POKE beispielsweise erlauben es, jede Speicheradresse auszulesen oder zu beschreiben, und INP und OUT greifen direkt auf die Hardware-Ports zu.

Ein weiterer wesentlicher Vorteil von Basic besteht darin, daß es Typumwandlungen automatisch erledigt. Das heißt, man kann jederzeit eine Integer-Zahl mit einer Double-Precision-Variable multiplizieren, und Basic erledigt die Umrechnung automatisch. Die Stringbearbeitung ist ebenso mächtig; es gibt Funktionen, die Zeichen an jede Stelle schreiben oder von jeder Stelle holen. Ein weiteres mächtiges Feature ist die Grafik. Basic hat eingebaute Funktionen zum Zeichnen von Kreisen, Rechtecken, gefüllten Rechtecken usw. auf alle gängigen Monitortypen.

Warum hat sich Crescent Software auf die Entwicklung einer Bibliothek von Programmiertools für Basic konzentriert?

Bei den Programmiersprachen gab es in den letzten paar Jahren zweifellos gewaltige Fortschritte. Eine Sprache aber kann noch so vollständig sein und noch so gut konzipiert, die Programmierer werden immer meinen, daß sie dieses und jenes noch brauchen. Traditionell füllen Bibliotheken – oder Toolboxen – von unabhängigen Herstellern diese Lücke. Eine Sprache wie C braucht wegen ihrer Beschränktheit die Unterstützung externer Bibliotheken. »Reines« C kann beispielsweise weder den Bildschirm löschen noch den Cursor positionieren, so daß C-Programmierer Toolbox-Routinen verwenden müssen. Meist glauben sie, das müsse so sein. Auch Pascal hat seine spezifischen Einschränkungen und mittlerweile sind Toolboxen für diese Sprache ebenso alltäglich.

Der vielleicht wichtigste Grund, eine externe Bibliothek zu verwenden, ist die Reduzierung des Entwicklungsaufwands für ein Programm. Natürlich kann ein erfahrener Programmierer ein Pull-Down-Menüsystem oder einen voll ausgebauten Editor samt Mausunterstützung entwickeln, aber das kostet enorm viel Zeit, Zeit die man besser in den Rest des Programms investiert. Dazu sind die Anwender an hübsche Bildschirmausgaben gewöhnt, an Pop-Up-Windows und was sonst noch zu einer ausgefeilten Benutzeroberfläche gehört. Eine gute Toolbox liefert dafür Lösungen »von der Stange« und verbessert und erweitert gleichzeitig die Programmiersprache.

Hat sich die Bibliothek einfach aus einer Sammlung von Utilities ergeben, die Sie im Laufe der Zeit aufgebaut haben, oder haben Sie sich eigens daran gemacht, sie zu schreiben?

Ein Sortiment von Tools zu entwerfen erfordert wesentlich mehr Aufwand als, sagen wir, eine Schnittstelle zu den verschiedenen DOS- und BIOS-Funktionen zu schreiben. Oft fangen wir an, ein Programm zu entwerfen und entdekken als erstes, daß wir dazu eine oder mehrere spezielle Routinen brauchen. Häufig führte unser eigener Bedarf zur Entwicklung von Funktionen oder Unterprogrammen, die sich dann auch in einem weiteren Rahmen als nützlich erwiesen.

Ein Beispiel sind die Funktionen, die das Minimum und das Maximum zweier Werte liefern. Sie werden in vielen Programmen von QuickPak Professional häufig benutzt und vermeiden eine Menge von IF/THEN-Anweisungen. Die MinInt-Assembler-Funktion liefert das Minimum zweier Integer-Variablen und erspart Sequenzen wie

```
IF X > MaxAllowed THEN
Value = MaxAllowed
ELSE
Value = X
END IF
```

Das gleiche erledigt jetzt eine Anweisung:

```
Value = MinInt (X, MaxAllowed)
```

Zur Unterstützung bei der Entwicklung unseres Editors QEdit haben wir eine Menge weiterer Funktionen geschaffen, die ebenfalls zu sinnvollen Ergänzungen für das Paket wurden.

Als wir QuickPak Professional konzipierten, setzten wir uns mehrere sehr klare Ziele. Wir wollten fertige Lösungen für allgemeine Programmierprobleme anbieten. Sie lassen sich in zwei Klassen unterteilen: Routinen, die für die meisten Programmierer zu schwierig sind, um sie selbst zu schreiben, und Operationen, die mit Basic alleine nicht möglich sind.

Ein Beispiel für die erste Klasse sind Assembler-Routinen, etwa um ein Array schnell zu durchsuchen oder zu sortieren. Andere schwierige Sachen wären ein Texteditor, ein Spreadsheet oder eine Stoppuhr mit Mikrosekunden-Auflösung.

Die zweite Kategorie besteht aus DOS- und BIOS-Interrupts, die Basic nicht so ohne weiteres ansprechen kann. Man braucht sie, um die Dateien auf der Platte aufzulisten, ihr Datum und ihre Erstellungszeit zu ändern oder um ein Plattenlabel zu lesen oder zu schreiben.

Zweitens wollten wir, daß diese Funktionen sehr einfach einzusetzen sind. Beispielsweise gibt es von allen Routinen, die Strings durchsuchen und bearbeiten, eine Version, die Groß- und Kleinschreibung unterscheidet und eine, die das nicht tut. Wichtig war auch, die Zahl der Übergabeparameter auf ein absolutes Minimum zu beschränken, und die Routinen möglichst als Funktionen zu implementieren. Funktionen sind, anders als Unterprogramme, die natürlichste Art, Basic zu erweitern.

Drittens sind in einigen Fällen die Fehlerbehandlungsmöglichkeiten von Basic nicht ausreichend. In C oder Pascal ist es möglich, eine Datei zu öffnen und dann nachzuprüfen, ob ein Fehler aufgetreten ist. Basic dagegen verlangt eine spezielle vorab definierte Fehlerroutine. Wenn ein Fehler auftritt, landet man in der Fehlerroutine – oft ohne daß man eine Ahnung hat, wie oder woher man dorthin gekommen ist. Hier sind Möglichkeiten, Laufwerk oder Drucker zu überprüfen, oder die Fehlerroutine von Basic ganz zu umgehen, nützliche Ergänzungen.

So wie der Sourcecode und die mitgelieferten Tutorials aussehen, könnte man meinen, daß Sie Basic lehren wollen.

Sicher. Oft brauchen Programmierer weniger neue Sprachfeatures als vielmehr ein Verständnis dafür, wie man die bereits vorhandenen Features nutzt. Das absolut nützlichste Tool eines Programmierers ist Wissen.

Jeder erfahrene Programmierer wird Ihnen bestätigen, daß der beste Weg zum Könner darin besteht, die Programme anderer zu studieren. Tatsächlich sind viele Programmierer Autodidakten, die ausschließlich aus Büchern und Zeitschriftenartikeln lernen. Wenn man das Programm eines anderen nimmt, den Sourcecode studiert und eventuell verändert, entsteht ein viel tieferes Verständnis.

Wir wollen, daß die Leute verstehen, wie diese Routinen arbeiten, und daß sie davon lernen können. Das bedeutet nicht nur, daß man sämtlichen Basic- und Assemblercode verfügbar macht, sondern auch diverse Tutorials schreiben muß, die die dahinterstehenden Konzepte erklären.

Die Handbücher von QuickBasic 4 sind, so weit sie gehen, ausgezeichnet, aber zu einer ganzen Anzahl wichtiger Punkte sagen sie gar nichts, beispielsweise zur Übergabe von TYPE-Variablen und Arrays an Unterprogramme und Funktionen. Die Beispiele, in denen TYPE-Parameter verwendet werden, kaschieren das Problem, indem sie das Array als SHARED deklarieren. Ebenso steht im Handbuch kein Wort zum Sichern und Laden von Arrays sowie EGA-Bildschirmdumps auf und von der Platte.

Programmierer, die ihr Basic verbessern wollen, müssen diese Konzepte verstehen. QuickPak Professional enthält viele solcher Informationen, zusammen mit Tutorials zu den Dateizugriffen oder zum Unterbringen von Strings und Zahlen im Codesegment des Programms, und einen Vergleich der verschiedenen Arten des Prozedurdesigns.

Eines der angenehmen Dinge bei Basic ist, daß es den Programmierer davon befreit, jede Einzelheit über das Betriebssystem wissen zu müssen. Das führt allerdings auch zu gewissen Beschränkungen. Können Sie einige Beschränkungen nennen, die Sie überwunden haben ohne die Sprache dadurch komplizierter zu machen?

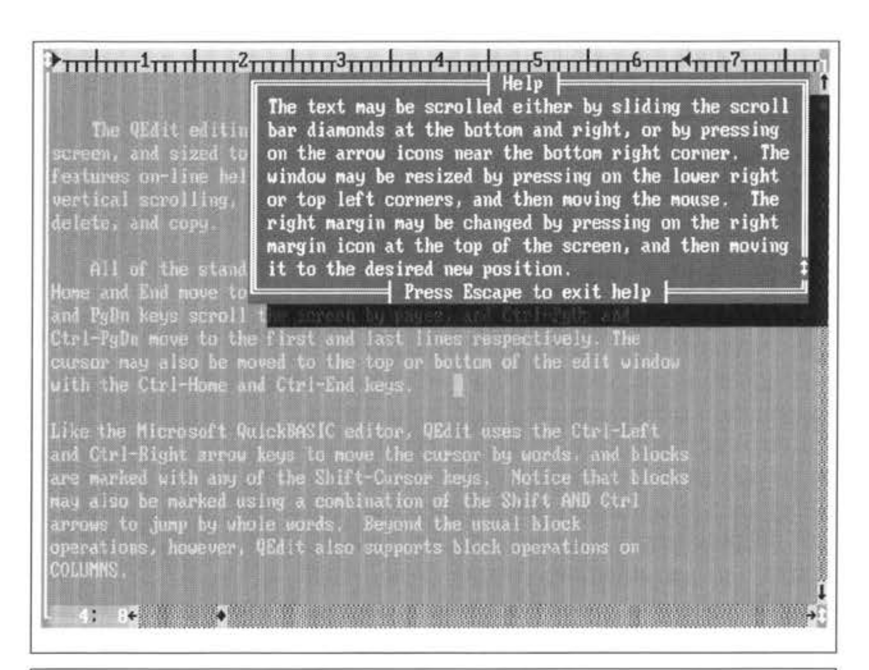
Basic hat viele eingebaute Kommandos, aber es fehlen Möglichkeiten zum Ansprechen der DOS- und BIOS-Interrupts. QuickBasic 2 brachte so etwas mit CALL INTERRUPT, allerdings war es ein Zusatz, der nur über ein zugebundenes Objekt-Modul zur Verfügung stand. Dazu ist CALL INTERRUPT umständlich einzusetzen und man muß über die DOS-Funktionen Bescheid wissen, was bei vielen Basic-Programmierern nicht der Fall ist.

Statt nun einfach nur einen etwas verbesserten Ersatz für CALL INTERRUPT bereitzustellen, gingen wir davon aus, daß ein Basic-Programmierer auf keinen Fall wissen muß, wie die DOS- und BIOS-Funktionen angesprochen werden.

Lassen Sie mich ein Beispiel geben. Jede DOS-Funktion, die einen Dateinamen entgegennimmt, erwartet diesen Namen im ASCIIZ-Format, so wie Strings in C abgelegt werden. In Basic jedoch haben Strings kein Nullbyte als Endemarkierung. Statt dessen gibt es einen Stringdeskriptor für jeden String oder jedes Stringarray-Element. Ein Stringdeskriptor ist eine 4-Byte-Tabelle, die die Länge und Adresse des Strings enthält.

Um CALL INTERRUPT zu benutzen, muß der Basic-Programmierer jedes Mal, wenn er einen Dateinamen an eine DOS-Funktion übergeben will, das Nullbyte selbst hinzufügen. Weil einfache Anwendung für uns höchste Priorität hatte, entschieden wir uns dafür, ihn den Dateinamen in ein Zwischenfeld kopieren zu lassen, und die Routine fügt das abschließende Nullbyte an, bevor sie DOS aufruft.

Eine wichtige DOS-Funktion, die von den meisten Programmen benötigt wird, ist die Möglich-



READY

A20 'Barient Winches #7890B

	A	B	C	D	E	F
1	This the title line of a sample spreadsheet					
2	===					
3						
4	Part Description	Quantity	Units	Unit Price	Total Value	
5	----------------------	--------	-------	----------	-----------	
6	Deck Fill Asy.	26	Pcs.	$12.34	$320.84	
7	Exhaust Hose 3 in.	25	Feet	$2.51	$62.75	
8	Babbit Disk 8 in.	5	Pcs.	$85.00	$425.00	
9	Shaft Coupler 1.25 in.	5	Pcs.	$54.23	$271.15	
10	Prop. Shaft 1.25 in.	3	Pcs.	$250.00	$750.00	
11	Stuffing Box 1.25 in.	20	Pcs.	$56.23	$1,124.60	
12	Cutlass Bearing 1.25 in.	12	Pcs.	$23.76	$285.12	
13	Shaft Zincs 1.25 in.	32	Pcs.	$13.42	$429.44	
14	Beckson Deck Plates	5	Pcs.	$29.89	$149.45	
15	Par Elec. Bilge Pump	2	Pcs.	$165.99	$331.98	
16	Seacocks	4	Pcs.	$231.00	$924.00	
17	Heat Exchanger	1	Pcs.	$287.23	$287.23	
18	Beckson Air Ports	8	Pcs.	$76.34	$610.72	
19	Mariman Snatch Blocks	7	Pcs.	$165.23	$1,156.61	
20	Barient Winches #7890B	2	Pcs.	$803.23	$1,606.46	
21	Barient Winches #7120A	2	Pcs.	$756.89	$1,513.78	

Dieser Editor und das Spreadsheet wurden in Basic und mit Hilfe der Funktionen von Crescents Bibliothek entwickelt.

keit, eine Liste der Dateien auf der Platte zu bekommen. Keine Anwendung, die etwas taugt, wird den Anwender zwingen, die Namen der zu ladenden Dateien im Kopf zu haben. Statt dessen sollte das Programm alle vorhanden Dateien auflisten und irgendeine Methode bieten, eine davon auszuwählen.

In DOS gibt es keine Möglichkeit, mit einer einzigen Operation eine vollständige Liste der Dateinamen zu bekommen. Sogar wenn es sie gäbe, müßte das Basic-Programm vorher die Anzahl der Namen wissen, damit es ausreichend Speicher dafür reservieren kann. Unsere Lösung besteht darin, daß wir eine Funktion schrieben, die die Zahl der Dateinamen zurückgibt, die einem gegebenen Suchkriterium genügen. Sobald diese bekannt ist, kann ein passendes String-Array reserviert werden, und eine zweites Unterprogramm holt dann sämtliche Namen auf einmal.

Damit sind wir an einem interessanten Punkt. In Microsoft Basic gibt es zwei Arten von Unterprogrammen: Unterprogramme, die per CALL, und Funktionen, die über einen Basic-Ausdruck aufgerufen werden. Sie scheinen Funktionen zu bevorzugen.

Funktionen sind wirklich praktisch und ein echter Gewinn. Man kann eine Basic-Funktion in Basic oder in Assembler schreiben. Im Mixed-Language Programming Guide für den Macro Assembler (MASM) Version 5 steht, wie man Assembler-Funktionen für QuickBasic implementiert.

Ein wesentlicher Vorteil von Funktionen ist die Eliminierung eines Übergabeparameters. Wenn die Routine, die die Zahl der gefundenen Dateinamen liefert, eine Prozedur wäre, müßte sie folgendermaßen aufgerufen werden:

```
CALL FCount ("*.*", Count)
PRINT Count; " Dateien wurden gefunden"
```

Dieselbe Routine als Funktion ist einfacher anzuwenden und zu verstehen:

```
PRINT FCount ("*:*");_
        " Dateien wurden gefunden"
```

Das Ergebnis einer Basic-Funktion kann direkt in einer PRINT-Anweisung benutzt werden, oder, wie beim Reservieren der erforderlichen Stringelemente, als Teil des DIM-Kommandos:

```
DIM Array$ (FCount ("*.*"))
```

Funktionen wie diese sind der natürlichste Weg, die Sprache Basic zu erweitern, und weil sie einen Parameter weniger benötigen, ist eine Funktion auch schneller als eine entsprechende Prozedur.

Parameter werden in allen Microsoft-Sprachen auf dem Stack übergeben, und es ist kein Geheimnis, daß Stackoperationen ziemlich langsam sein können. Deshalb sind Funktionen besonders gut, wenn sie ohne Parameter eingesetzt werden. Wir haben beispielsweise eine Anzahl Funktionen, die den Status verschiedener Tastatureinstellungen zurückgeben. Die übliche Methode, den Status, sagen wir, der Alt-Taste festzustellen, geschieht mit Hilfe der DEF SEG-, PEEK- und AND-Kommandos von Basic:

```
DEF SEG = 0
AltKey = PEEK (&H417) AND 8
IF AltKey THEN PRINT_
        "Die »Alt«-Taste ist gedrückt"
```

Mit einer speziellen Funktion kann man diesen ganzen Code durch eine einzige Anweisung ersetzen:

```
IF AltKey THEN PRINT_
        "Die »Alt«-Taste ist gedrückt"
```

Ohne Parameterübergabe braucht man zur Implementierung einer solchen Funktion nur fünf Assembler-Anweisungen (*Listing 1*).

Wieder taucht Assembler auf. Es scheint, als wäre es sogar für Basic-Programmierer nötig, Assembler zu können. Was glauben Sie?

Viele Hochsprachen-Programmierer würden gerne Assembler lernen, schrecken aber davor zurück, weil sie fürchten, daß das langwierig und

mühsam ist. Doch Assembler zu lernen, ist gar nicht so mühsam, und schon ein bißchen darüber zu wissen, ist für einen Basic-Programmierer nützlich. Um eine Assembler-Routine anzuwenden, braucht man natürlich keine Assembler-Kenntnisse.

```
Xor  AX,AX           ;look at BIOS data in low memory
Mov  ES,AX
Mov  AL,ES:[417h]    ;get the keyboard status byte
And  AL,00001000b    ;clear all but the Alt Key bit
Ret                  ;return to BASIC, AX holds result
```

Listing 1: Ein Beispiel für eine Basic-Funktion, die mit MASM geschrieben wurde.

Muß man bei Assembler-Funktionen etwas besonderes beachten?

Was einem als erstes einfällt, ist, daß eine Assemblerfunktion, bevor sie in Microsoft Basic verwendet werden kann, in dem Programm, das sie verwenden will, deklariert werden muß. Daß man Prozeduren deklarieren muß, ist neu in Basic, aber in diesem Fall durchaus sinnvoll. Andernfalls würde QuickBasic in dem Beispiel mit der Alt-Taste annehmen, daß »AltKey« einfach eine Integervariable ist. Wenn man sie vorher deklariert, weiß QuickBasic, daß »AltKey« eine aufzurufende Funktion ist und daß der Wert im AX-Register ihr Ergebnis darstellt.

Und dann macht Assembler gelegentlich die Sache auch etwas schneller.

Das ist ein ebenso wichtiger Grund, Assembler einzusetzen. Wir wollten diejenigen Operationen beschleunigen, die in allen Hochsprachen typischerweise sehr langsam sind. Assembler ist nun einmal der Schlüssel für eine gute Performance, und etwa zwei Drittel aller Routinen in QuickPak Professional sind in Assembler geschrieben.

Die meisten Compiler für DOS wickeln Bildschirmausgaben über das Betriebssystem ab, um mit möglichst vielen Rechnern kompatibel zu sein. Ob über DOS oder das BIOS, es dauert zu lange. QuickBasic 4 schafft hier etwas Abhilfe, indem es direkt in den Videospeicher schreibt, aber auch da gibt es noch viel zu verbessern. Jedes Zeichen wird darauf geprüft, ob es ein spezielles Kontrollzeichen ist, und bei jedem Zeichen geht nochmals Zeit für die Prüfung verloren, ob der Bildschirm gescrollt werden muß. Wir haben einige sehr schnelle Routinen für die Ausgabe auf mehrere Video-Seiten geschrieben und für eine Textausgabe, die die aktuellen Bildschirmfarben nicht zerstört.

Besonders nützlich ist eine Routine, die einen Teil eines String-Arrays ausgibt, indem es ihn direkt in einen angegebenen Bildschirmbereich schreibt. Das vereinfacht die Entwicklung beispielsweise einer Browse-Funktion, mit der man am Bildschirm durch ein ganzes Array scrollen kann. Auch hier war unser Ziel, daß diese Routinen so viel wie möglich selbst erledigen und dem Programmierer unnötige Arbeit ersparen.

Assemblerroutinen können dem Basic-Programmierer auch eine wesentlich schnellere Array-Verarbeitung bescheren.

Genau das ist es, wo Assembler Basic wirklich ein wesentliches Stück weiterbringt, bei der Arrayverarbeitung. Wir stellen beispielsweise Funktionen, die den größten oder kleinsten Wert liefern, für jeden Arraytyp in Microsoft Basic zur Verfügung. Eine spezielle Assemblerroutine ist gewöhnlich sechs bis sieben Mal schneller als eine entsprechende Basic-Funktion.

Eine noch größere Verbesserung läßt sich beim Speichern eines ganzen Stringarrays auf die Platte und beim späteren Wiedereinlesen erreichen. Eine der langsamsten Basic-Operation ist das Einlesen von Daten aus einer sequentiellen Datei. Dabei muß jedes einzelne Byte geprüft werden, ob es ein Carriage Return ist, das ein Stringende markiert, oder ein CHR$(26), das das Dateiende anzeigt. Eine spezielle Assemblerroutine, die mit einer Operation die ganze Datei angeht, spart enorm Zeit.

Die schnellste Art, in Basic ein Zahlenarray abzuspeichern, ist BSAVE. Das erspart eine Menge von PRINT-Ausgaben in eine Datei. Deren Hauptproblem ist der Aufwand, der nötig ist, um die Werte vom internen PC-Format in die entsprechenden ASCII-Ziffern zu konvertieren. Obendrein braucht BSAVE weniger Plattenplatz.

Integerzahlen belegen im Hauptspeicher nur zwei Bytes. In der ASCII-Darstellung dagegen brauchen sie bis zu fünf Bytes, und bis zu sechs, wenn die Zahl negativ ist. Zusätzlich ist für jede Zahl in der Datei noch ein Carriage Return plus Line Feed oder ein trennendes Komma nötig. BSAVE dagegen macht einen »Schnappschuß« von einem beliebigen Bereich im Speicher und schickt ihn in einem Zug auf die Platte. Der komplementäre Befehl ist BLOAD, der eine zuvor geschriebene BSAVE-Datei wieder einliest.

Was ist mit Stringarrys?

Anders als Zahlenarrays erstrecken sich Stringarrays in Basic nicht über zusammenhängende Speicheradressen. Statt dessen liegen die Deskriptoren-Tabellen hintereinander, und die tatsächlichen Daten können irgendwo im aktuellen Segment sein. Deshalb müssen die Daten aller Elemente in einem einzigen Block gesammelt werden, ehe BSAVE auf einen Stringarray angewendet werden kann. Zwei spezielle Routinen bearbeiten sämtliche Stringelemente und kopieren sie in ein Integerarray. Eine zusätzliche Routine erlaubt, wenn nötig, ein einzelnes Stringelement einzulesen.

Zahlenarrays sind bei QuickBasic nicht auf das aktuelle Segment begrenzt. Allein schon ein Stringarray in ein anderes Segment kopieren zu können ist eine nützliche Sache. Und wenn der Platz knapp wird, kann man die Arrays leicht auf Platte auslagern und so Platz gewinnen.

Listing 2: WORDWRAP.BAS

```
DEFINT A-Z
DECLARE SUB WordWrap (X$, Wide)

CLS

A$ = "BASIC strings use a descriptor table ot tell each string's"
B$ = "length and memory location. At first glance, it seems that"
C$ = "any two strings can be exchanged by just swapping their"
D$ = "descriptor tables. Once we could exchange strings for"
E$ = "sorting purposes, it was a simple matter to insert or"
F$ = "delete elements in an array. On a standard IBM PC, inserting"
G$ = "a single string at the beginning of a 2000 element array"
H$ = "takes more thatn two seconds using Microsoft QuickBASIC."

W$ = A$ + B$ + C$ + D$ + E$ + F$ + G$ + H$
PRINT W$
PRINT
Wide = 60                'the maximum width of the display
WordWrap W$, Wide


SUB WordWrap (X$, Wide%)

  Length% = LEN(X$)            'remember the length
  Pointer% = 1                 'start at the beginning of the string

  'scan a block of sixty characters backwards, looking for a blank
  'stop at the first blank, or if we reached the end of the string
   DO
     FOR X% = Pointer% + Wide% TO Pointer% STEP -1
        IF MID$(X$, X%, 1) = " " OR X% = Length% + 1 THEN
         'LOCATE , LeftMargin   'optional to tab in the left edge
          PRINT MID$(X$, Pointer%, X% - Pointer%);
         'LPRINT [TAB(LeftMargin)]; MID$(X$, Pointer%, X% - Pointer%)
          Pointer% = X% + 1
          WHILE MID$(X$, Pointer%, 1) = " "
           Pointer% = Pointer% + 1 'swallow extra blanks to next word
          WEND
          IF POS(0) > 1 THEN PRINT  'if cursor didn't wrap next line
              EXIT FOR                  'done with this block
           END IF
       NEXT
    LOOP WHILE Pointer% < Length%       'loop until done

END SUB
```

Weil wir zusätzlich jeden String mit einem Carriage Return/Line Feed abschließen, kann die Datei von jeder Anwendung direkt gelesen werden.

Sie haben den Basic-Benutzern zwei »neue« Datentypen gegeben. Was sind »sehr lange Integer« und »Bit Arrays«?

Aus der Arbeit an QuickPak Professional ergaben sich mehrere interessante Konzepte und Routinen, unter anderem zwei »neue« Datentypen.

Ein wichtiges und lange überfälliges Feature von QuickBasic 4 ist die Unterstützung langer (4 Byte) Integers. Es gibt Fälle, in denen lange Integers entschieden vorteilhafter sind als Gleitkommazahlen, weil man sie sehr schnell und ohne Rundungsfehler berarbeiten kann. Buchhalter mögen das. Wenn man sie für Pfennige nimmt, ist ihr Wertebereich von +/- DM 21.474.836,47 im allgemeinen ausreichend – außer für ernsthafte Finanzgeschäfte.

Als Lösung haben wir eine Reihe von Routinen entworfen, um, wie wir es nennen, sehr lange Integer zu addieren, subtrahieren, multiplizieren und dividieren. Diese Variablen sind acht Byte lang und erlauben die Verarbeitung extrem großer Zahlen ohne Rundungsfehler. Man definiert eine »Very Long«-Variable, indem man sie einer gewöhnlichen »Double Precision«-Variable als Alias-Namen zuweist und Basic merkt nicht das geringste davon.

Andererseits haben wir zwei Routinen für die Behandlung von Bit-Arrays erstellt. Die kleinste Variable, die Basic bietet, ist eine 16-Bit-Integer-Variable. Wenn ein Programm den Wertebereich von Integern braucht, ist es absolut sinnvoll, damit zu arbeiten. Aber oft braucht man nicht mehr als einfache Ja/Nein-Variablen und ein Integerarray verschwendet dann enorm viel Speicher. Ein Bit-Array mit 1000 Elementen belegt nur 125 Byte, während ein gewöhnliches Integer-Array 2000 Byte braucht.

Sie haben auch eine Routine, mit der Basic-Programme gleichzeitig auf zwei Monitore schreiben können. Wie haben Sie das hingekriegt?

Es ist kein Problem, eine Routine zu machen, die den aktiven Monitor wechselt, wobei jedes Umschalten zwischen den Monitoren bedeutet, daß der Bildschirm gelöscht wird. Weil nichts das Programm daran hindern kann, direkt in den Bildschirmspeicher eines inaktiven Monitors zu schreiben, konnten wir unsere Lösung elegant implementieren.

Alle Video-Routinen von QuickPak Professional stellen bei ihrem Aufruf automatisch den Typ des installierten Monitors fest. Das ist aus zwei Gründen nötig. Zum einen ist das Videosegment bei Farb- und Monochrom-Monitoren unterschiedlich, so daß das richtige Segment bekannt sein muß, damit die Routine direkt in den Bildschirmspeicher schreiben kann. Zum andern, und das ist ebenso wichtig, »schneien« CGA-Bildschirme, wenn man Lesen und Schreiben nicht mit der Bild-Wiederholfrequenz synchronisiert.

In der Routine, die auf jeden beliebigen Monitor schreibt, muß der Aufrufer den Monitor, auf den geschrieben werden soll, mit einem Code angeben; 1 bedeutet Monochrom, 2 ist ein CGA- und 3 steht für einen EGA- oder VGA-Schirm. Obwohl ein PC zwei Monitore gleichzeitig unterstützen kann, müssen sie unterschiedlich sein – das heißt, sie können nicht beide ein Farb- oder Monochrom-Monitor sein. Deshalb braucht der Programmierer nur den Monitortyp anzugeben, der nicht der aktuelle ist. Natürlich gibt es auch eine Routine, die den Typ des gegenwärtig aktiven Monitors liefert.

Sie bieten auch eine Möglichkeit, Bildschirm-Dumps unabhängig vom Videomodus zu machen. Was können Sie uns dazu sagen?

Wir wollten auch Grafik angehen. Eine wesentliche Einschränkung des GRAPHICS-Programms von DOS ist, daß es nur mit den CGA-Modi funktioniert. Dazu funktioniert es nur mit Druckern, die mit dem Epson-/IBM-Befehlssatz kompatibel sind. Weil Basic EGA, VGA und Hercules unterstützt, war es wichtig, eine Routine bereitzustellen, die Grafiken aus diesen Modi heraus drucken läßt.

Wir fanden, daß eine wirklich nützliche Print-Screen-Routine nicht nur die Steuerzeichen der

Epson-Drucker beherrschen muß (die alle modernen Drucker emulieren können), sondern auch die des HP Laserjets. Und weil ein Laserjet in mehreren Größen drucken kann, sollte der Aufrufer die gewünschte Größe angeben können. Dann war nur noch zu klären, was mit den Farben geschehen soll. Ein noch so beeindruckendes dreidimensionales Diagramm in sechzehn Farben ist als Hardcopy nutzlos; alles was man bekommt, ist ein großer schwarzer Kreis. Die eindeutig beste Lösung ist es, jede der möglichen Bildschirmfarben durch ein Schraffurmuster zu ersetzen. Was die Sache allerdings ausgesprochen kompliziert macht, sind die verschiedenen Arten, in denen der Grafik-Bildschirmspeicher organisiert ist.

Im Textmodus befinden sich aufeinanderfolgende Zeichen in aufeinanderfolgenden Bytes im Bildschirmspeicher. In den CGA-Modi dagegen ist der Bildschirmspeicher mit einer Methode namens »Interlacing« organisiert. Bei einem solchen System ist jede zweite Zeile in einem anderen Speicherblock, und das macht die Speicherzugriffe um vieles schwieriger.

Auch Hercules arbeitet mit Interlacing, nur ist es da jede vierte Zeile. EGA und VGA arbeiten wieder nach einem anderen Prinzip, bei dem jede Farbe in einem eigenen Segment steht. Das erschwert es, den Bildschirmspeicher zu lesen oder Farben in Schraffurmuster umzusetzen.

Listing 3: GETCOLOR.BAS

```
DEFINT A-Z
DECLARE SUB GetColor (FG, BG)             'gets BASIC's current colors
DECLARE SUB SplitColor (XColor, FG, BG) '.ASM - splits a color into FG
                                          ' and BG
DECLARE FUNCTION OneColor% (FG, BG)      '.ASM - combines FG/BG into one
                                          ' color


CLS
INPUT "Enter a foreground color value (0 to 31): ", FG
INPUT "Enter a background color value (0 to 7) : ", BG
COLOR FG, BG

PRINT : PRINT "BASIC's current color settings are: ";
GetColor FG, BG
PRINT FG; "and"; BG

PRINT "That combines to the single byte value of"; OneColor%(FG, BG)
PRINT "Broken back out results in";
SplitColor OneColor%(FG, BG), NewFG, NewBG
PRINT NewFG; "and"; NewBG

COLOR 7, 0      'restore defaults before ending

'This function obtains BASIC's current colors by first saving the
'character and color in the upper left corner of the screen.  Next,
'a blank space is printed there, and SCREEN is used to see what color
'was used.  Finally, the original screen contents are restored.
'
SUB GetColor (FG%, BG%) STATIC
  V% = CSRLIN                        'save the current cursor location
  H% = POS(0)
  SaveChar% = SCREEN(1, 1)           'save the current character
  SaveColor% = SCREEN(1, 1, 1)       'and its color
  SplitColor SaveColor%, SaveFG%, SaveBG%

  LOCATE 1, 1                        'print with BASICƒs current color
  PRINT π π; CHR$(29);               'back up the cursor to 1,1
  CurColor% = SCREEN(1, 1, 1)        'read the current color
  COLOR SaveFG%, SaveBG%             'restore the original color at 1,1
  PRINT CHR$(SaveChar%);             'and the character

  LOCATE V%, H%                      'put the cursor back where it was
  SplitColor CurColor%, FG%, BG%     'split the color into separate
                                     'FG and BG
  COLOR FG%, BG%                 'restore BASIC's current value for it
END SUB
```

Eine ihrer Leistungen war die Entwicklung von QEdit, einem in Basic geschriebenen Textprogramm. Die variabel langen Strings von Basic müssen dabei sehr nützlich gewesen sein.

Sprachen wie C oder Pascal, die nur Strings mit fester Länge zulassen, erfordern viel mehr Programmieraufwand, um zu den gleichen Ergebnissen zu kommen. Das Programm vergeudet entweder den größten Teil des Speichers für Textzeilen, die kürzer als das Maximum sind, oder es muß eine Pointertabelle mit den Adressen führen, an denen die einzelnen Strings im Speicher stehen. Immer wenn man Zeichen und Zeilen einfügt oder löscht, müssen die Pointer aktualisiert werden.

Die Tatsache, daß Basic variabel lange Strings unterstützt, hat uns sehr viel Arbeit erspart. Basic erledigt das alles sehr schnell und automatisch, und der Zeilenumbruch von QEdit kommt auch beim besten Schnellschreiber mit. (Siehe *Listing 2* für ein Beispiel des Zeilenumbruchs mit Basic und *Listing 3* für ein Beispiel einer Funktion zur Abfrage der Bildschirmfarben – Die Red.)

Variabel lange Strings müssen dynamisch allokiert werden. Das dürfte einige Probleme bereitet haben.

Mit am schwierigsten war die Entwicklung der Routinen zur Behandlung von Stringarrays. Dazu waren weitere Nachforschungen nötig. Vor allem brauchten wir einiges an Informationen über die internen Operationen von Microsoft Basic.

Weil Basic Strings von nahezu jeder Länge zuläßt, werden sie, wie Sie schon sagten, je nach Bedarf dynamisch allokiert. Das macht die Speicherverwaltung ausgesprochen schwierig, und verständlicherweise betrachtet Microsoft viele der Einzelheiten als sein Eigentum.

Das zeigte sich sofort, als wir versuchten, eine Routine zu schreiben, die ein Stringarray sortiert. Ich sagte bereits, daß Basic Strings über eine Beschreibungstabelle verwaltet, in der die Länge und die Speicheradresse eines jeden Strings steht. Auf den ersten Blick könnte man meinen, daß es für das Auswechseln zweier Strings reicht, ihre Beschreibungstabellen auszutauschen. Die Tabelleneinträge sind in den QuickBasic-Handbüchern gut dokumentiert, aber leider ist das nicht alles.

Was nicht erwähnt wird, ist, wie die Stringdaten verknüpft sind. Es dauerte einige Tage, bis wir es durch Probieren herausbekamen. Als wir dann Strings zum Sortieren vertauschen konnten, war es nur noch eine Kleinigkeit, Elemente einzufügen oder zu löschen. Das Einfügen eines einzigen Strings am Anfang eines Arrays mit 2000 Elementen dauert mit QuickBasic auf einem Standard-PC länger als zwei Sekunden. Eine entsprechende Assembler-Routine dagegen braucht nur eine Zehntelsekunde.

Einige der Stringfunktionen von QuickPak müssen ähnliche Probleme bereitet haben. Aber auch hier

scheinen Sie sich das Leben erleichtert zu haben, indem Sie daraus »Funktionen« machten.

Stringfunktionen waren tatsächlich einer der anderen Problempunkte – und wieder, weil die Informationen fehlten. Abgesehen davon, daß sie einen Übergabeparameter einspart, erspart eine Funktion, die einen String direkt übergeben kann, es dem Programmierer, zuvor immer erst Platz für sie zu reservieren.

Eine der Routinen in QuickPak Professional ermittelt das aktuelle Verzeichnis eines angegebenen Laufwerks. Weil ein Verzeichnisname bis zu 64 Zeichen lang sein kann, müßte eine solche Routine zuerst einen String dieser Länge allokieren. Am Ende müßten dann die überzähligen Zeichen von Hand entfernt werden.

Weil DOS alle Strings, die es liefert, mit einem Nullbyte abschließt, muß so ein Programm mit Basics INSTR-Funktion dieses Byte suchen; die Zeichen bis dahin übernimmt es dann.

```
Dir$ = SPACE$ (64)
          ' 64 Byte reservieren
Drive$ = "C"
          ' Laufwerk angeben
CALL GetDir (Drive$, Dir$)
          ' GetDir versorgt Dir$
Zero = INSTR (Dir$, 0)
          ' Nullbyte suchen
Dir$ = LEFT$ (Dir$, Zero - 1)
          ' alles vor der 0 aufheben
```

Als Stringfunktion ist GetDir wesentlich einfacher anzuwenden:

```
Drive$ = "C"
Dir$ = GetDir$ (Drive$)        oder
Dir$ = GetDir$ ("A")
```

Als Funktion sucht GetDir das Nullbyte und liefert dann einen String mit genau der richtigen Länge.

Kommen wir auf QEdit zurück. Offensichtlich haben Sie in QEdit die Editierkommandos von QuickBasic 4 emuliert. Es gibt schon so viele Editoren, warum haben Sie einen weiteren geschrieben?

Es gibt eine Menge Editoren zu kaufen, aber ich weiß keinen, bei dem der Sourcecode gratis mitgeliefert wird, und auch keinen, der dazu gedacht ist, in andere Programme eingebaut zu werden. Die Leute drängen uns oft, irgendwelche neuen Routinen zu entwickeln, aber am meisten fordern sie einen Texteditor, den sie anpassen und in ihre eigenen Programme einbauen können. Natürlich stellt sich unter einem »Editor« jeder etwas anderes vor und jeder Programmierer hat seine eigenen Vorstellungen, wie er arbeiten sollte.

Unsere Kunden sind mit den QuickBasic-4-Kommandos vertraut, und so war es naheliegend, diese zu emulieren. Aber weil der Sourcecode mitgeliefert wird, ist eine Anpassung an beliebige Tastenkombinationen jederzeit möglich. Aber die Editierkommandos festzulegen, war nur der Anfang.

Um wirklich nützlich zu sein, muß ein Texteditor Text als einzelne Zeilen akzeptieren und auch als »eine Zeile pro Absatz«, wie viele Textprogramme. Dazu sollte der Benutzer den Druckrand einstellen, die Fenstergröße verändern und den Text mit einer Maus durchlaufen lassen können – ohne daß das aufrufende Programm irgendetwas dazu tun müßte. Und selbstverständlich müssen die Tastatureingaben so schnell verarbeitet werden, daß sie für den Benutzer völlig transparent sind. QEdit hat einige wichtige Features wie Mausunterstützung, Blockoperationen und Online-Hilfe. Das vielleicht wichtigste Feature jedoch ist die Möglichkeit, nicht benötigte Features zu entfernen. Wenn jemand einen Minimaleditor will, nur mit Druckrand und Zeilenumbruch, dann sollte der Code für die Blockoperationen und die Mausunterstützung herausgenommen werden können. Wir haben diese Teile von QEdit deshalb in separate Abschnitte gelegt, wo der Anwender sie leicht löschen oder auskommentieren kann.

Ein weiteres Feature, das wir für unseren eigenen Bedarf brauchten, waren Blockoperationen für Spalten, für Worte und für Zeilen. Die meisten Editoren arbeiten nur im Satzmodus, so daß beim Markieren eines Blocks nach unten immer die ganze Zeile mitgenommen wird. Für ein Textprogramm ist das meist ausreichend. Aber wenn man programmiert, ist es sehr praktisch, wenn man einen langen Abschnitt markieren und seine Einrückstufe in einem verändern kann. Ein Spaltenmodus erleichtert auch das Kopieren oder Verschieben von Tabellen.

QEdit wurde offensichtlich als Pop-Up-Utility entworfen.

Obwohl QEdit nicht als eigenständiges Textprogramm gedacht ist, kann man ihn doch so benutzen. Allerdings ist er dafür konzipiert, daß die Anwender ihn als eine Pop-Up-Utility in ein Basic-Programm einbauen und so zeigen, daß mit Basic auch anspruchsvollere Programme geschrieben werden können.

Welche Schwierigkeiten hatten sie dabei?

Weil QEdit so viel kann, gibt es für ihn vier Hilfsbildschirme, die über F1 aufgerufen werden. Der Hilfstext ist im Codesegment des Programms untergebracht und wird von einer speziellen Assemblerroutine geholt. Das spart an die 2 Kbyte Stringspeicher. Und weil die Anwender die Maus- und Blockoperationen entfernen können, wird in jedem dieser Abschnitte eine Variable verändert, so daß der Code für die Hilfefunktion weiß, welche Bildschirme gebraucht werden.

Die gleiche Methode wie beim Speichern des Hilfetextes außerhalb des Stringspeichers benutzten wir für das Cut-and-Paste-Clipboard. Andernfalls könnte Basic, wenn ein größeres Dokument geladen ist, beim Einlesen des Blocks leicht der Speicher ausgehen.

Damit sind wir an einem wichtigen Punkt. Es war schon schwer, eine Spalte einzulesen, die über mehrere Bildschirmseiten gehen kann. Sie zu löschen oder woanders einzufügen war noch schwieriger. Um zu vermeiden, daß der Textblock Stringspeicher wegnimmt, ist das Clipboard als Integerarray implementiert. Wir hatten bereits einen Stringmanager geschrieben, der ein Stringarray ganz oder teilweise in ein Integerarray und zurück kopiert, aber das Einlesen und Einfügen einer Spalte erforderten zwei weitere spezielle Assembler-Routinen.

Eine Routine kopiert Elemente eines Stringarrays in ein Integerarray, wobei sie jeweils an einem bestimmten Offset beginnt und nur eine bestimmte Anzahl Zeichen kopiert. Wenn eine Zeile nicht bis zum Ende der Spalte reicht, wird der Rest mit Leerzeichen aufgefüllt. Die zweite Routine holt die Bestandteile des Strings Stück für Stück aus dem Integer-Array, so daß die Zeilenumbruch-Funktion den Inhalt des Clipboards wieder in den Text einfügen kann.

Zusätzlich brauchten wir eine Routine zum Schließen eines zuvor gesicherten Bildschirmbereichs. Weil QEdit ein Pop-Up-Programm ist, sichert es den darunterliegenden Bildschirm automatisch. Wenn sich die Bildschirmgröße ändert, kann das kompliziert werden.

Sie mußten also Routinen zur Veränderung der Größe von Bildschirmfenstern schreiben. Wie arbeiten die?

Der Anwender kann jederzeit die Ecke links oben oder rechts unten mit der Maus anklicken und verschieben. Solche Fenster sind natürlich nichts Neues, aber wir wollten, daß sich das Fenster wirklich ändert, anstatt nur anzuzeigen, wie es nach dem Loslassen des Mausknopfs aussehen würde.

Bei einigen Programmen verursacht das Verändern der Fenstergröße ein starkes Flackern. In QEdit benutzen wir eine Routine, die nur den Teil des Bildschirms schließt, der von der Veränderung betroffen ist.

Wir haben noch eine Menge ähnlicher Routinen für QEdit und ganz allgemein für QuickPak Professional geschrieben. Zum Beispiel eine, die schnell ein Stringarray durchläuft, um die Zeilenzahl festzustellen. Eine andere hält den Mauszeiger innerhalb eines bestimmten Bildschirmbereichs. Eine weitere ermittelt den aktuellen Videomodus, so daß sich ein Programm automatisch auf 25, 43 oder 50 Zeilen einstellen kann.

Wie man sieht, ist eine Unmenge von Details nötig, um ein voll ausgebautes Textprogramm zu implementieren, gleich welche Sprache man benutzt. QuickBasic und der Microsoft Basic 6.0-Compiler sind ideal für alle Anwendungen, die umfangreiche Stringmanipulationen erfordern, ganz besonders aber sind sie für Textverarbeitungsprogramme geeignet.

Sie sind fest davon überzeugt, daß Basic eine »richtige« Programmiersprache ist?

Ich habe volles Vertrauen zu Basic als einer ernsthaften Anwendungssprache. Sie erlaubte Millionen von Menschen zu programmieren – echtes Programmieren. Es ist schade, daß viele sonst so gut informierte Programmierer Basic 6.0 und QuickBasic nur wegen des Basic-Interpreters von DOS für ungeeignet für »echte« Anwendungen halten. Abgesehen von ihrer außerordentlichen Geschwindigkeit bieten Microsofts Basic-Produkte auch Rekursion, strukturierten Code, strukturierte Daten und viele andere Features, die zu einer modernen professionellen Sprache gehören.

Kompatibel, aber besser:

Das neue QuickPascal von Microsoft

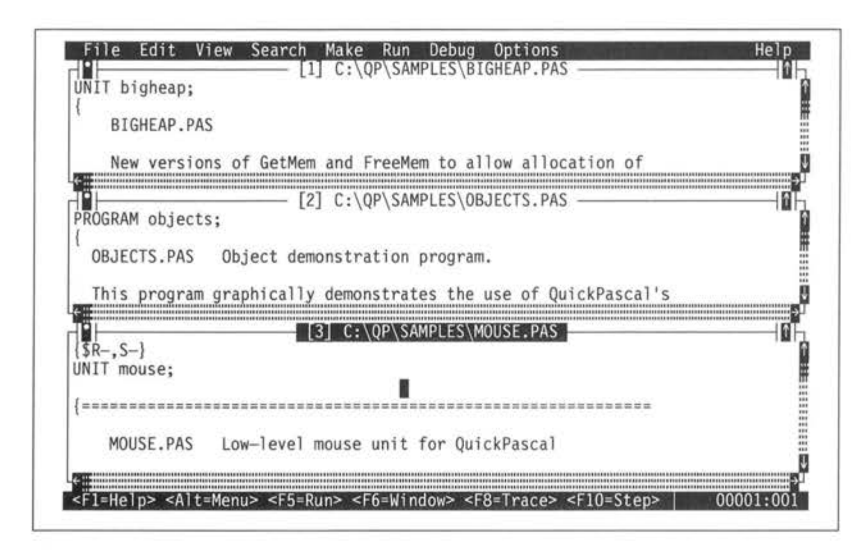


Bild 1: QuickPascal erlaubt die gleichzeitige Bearbeitung mehrerer Dateien in verschiedenen Bildschirmfenstern.

Mit QuickPascal wird die überaus erfolgreiche Quick-Sprachen-Reihe von Microsoft um einen leistungsstarken Pascal-Compiler erweitert, der sich gleichermaßen an Anfänger wie professionelle PC-Programmierer wendet. Dabei ist QuickPascal mehr als nur ein simpler Compiler, vereinigt es in sich doch eine komplette Entwicklungsumgebung mit Editor, Compiler, Linker, Debugger und allem, was sonst noch dazugehört. Beeindruckend sind aber auch die objektorientierten Leistungsmerkmale des neuen Compilers, die im folgenden Beitrag unter die Lupe genommen werden.

Wer Pascal sagt, der muß auch Turbo sagen, denn über Jahre hat dieser Compiler aus dem Hause Borland den Pascal-Markt unangefochten beherrscht. Dies hat dazu geführt, daß mittlerweile Millionen von Programmen existieren, die unter diesem Compiler entwickelt wurden. Niemand muß jedoch befürchten, daß dieser Entwicklungsaufwand umsonst war oder er sich größeren Portierungsproblemen gegenübersehen wird, sobald er sich für QuickPascal entscheidet, denn QuickPascal ist vollkommen »Source-kompatibel« zu Turbo Pascal, Version 5.0.

Auf den ersten Blick beweisen dies bereits die Demo-Programme von Turbo Pascal, wie etwa die kleine Tabellenkalkulation MCALC oder das Grafikdemo BGIDEMO, die ohne Veränderungen unter QuickPascal kompiliert und zum Laufen gebracht werden können. Daß QuickPascal aber noch einiges mehr zu bieten hat, als die Kompatibilität zu Turbo Pascal, das zeigt der folgende Testbericht.

Die Benutzeroberfläche von QuickPascal

Die verschiedenen Komponenten von QuickPascal – Editor, Compiler, Linker und Debugger – werden unter einer Benutzeroberfläche zusammengefaßt, die ganz den Richtlinien des SAA-Standards entspricht und dadurch den Bedienungskomfort realisiert, den dieser Standard verspricht. Im Mittelpunkt des Interesses stehen dabei das Hauptmenü, über das die Arbeit mit Editor, Compiler und Debugger gesteuert werden kann, sowie der Arbeitsbereich, der den weitaus größten Teil des Bildschirms einnimmt.

Hier – und bereits dies ist ein großer Vorteil gegenüber Turbo Pascal – kann der Anwender mehrere Fenster öffnen, um gleichzeitig verschiedene Quelldateien zu bearbeiten. Vor allem bei der Erstellung großer Programme, die sich aus mehreren Modulen zusammensetzen, erweist sich dies als sehr angenehm. Aber auch bei der Arbeit mit nur einer Quelldatei kann man aus dieser Möglichkeit Nutzen ziehen, denn dadurch lassen sich verschiedene Teile dieser Datei in getrennten Fenstern gleichzeitig auf dem Bildschirm betrachten und editieren.

Wer schon einmal permanent zwischen der gerade entwickelten Prozedur und dem Programmkopf hin- und herspringen mußte, um die exakte Schreibweise globaler Variablen oder Konstanten nachzusehen, wird dieses Feature zu schätzen wissen.

Die Lage und Größe der verschiedenen Fenster wird von QuickPascal übrigens nicht statisch vorgegeben, sondern kann vom Anwender festgelegt werden. Jederzeit können die verschiedenen Fenster vergrößert/verkleinert und verschoben werden. Auch ein kurzfristiges »Zoomen« des Fensters auf Bildschirmgröße ist mit

Hilfe eines einfachen Tastendrucks möglich und ebenso leicht kann es später wieder auf seine ursprüngliche Größe reduziert werden.

Alle diese Funktionen können übrigens auch über die Maus erreicht werden, die – dem SAA-Standard folgend – voll in die Bedienung der Benutzeroberfläche integriert wurde. Dies gilt nicht nur für die Arbeit innerhalb der verschiedenen Fenster, sondern gleichermaßen auch für die Aktivierung von Befehlen über das Hauptmenü und die Dateneingabe in Dialogboxen. Tatsächlich vollzieht sich beispielsweise das Vergrößern/Verkleinern bzw. Verschieben der Fenster im Arbeitsbereich mit Hilfe der Maus sogar komfortabler als mit Hilfe der Tastatur, da sie sich für derartige Aufgaben besser eignet.

Wie oben bereits erwähnt, können alle Befehle über das Hauptmenü erreicht werden, das jederzeit zur Verfügung steht. Um den Anfänger dabei nicht durch die Vielzahl der Befehle zu verwirren, kann der Anwender zwischen einer Anfänger- und einer Profi-Version von QuickPascal wählen. Zwar stehen alle Befehle auch in der Anfänger-Version weiterhin zur Verfügung, doch werden sie nicht mehr in den verschiedenen Menüs angezeigt und können daher auch nicht angewählt werden.

Schreitet der Anfänger in seinem Lernprozeß fort, kann er auf die Profi-Version umschalten, ohne das Programm verlassen zu müssen. Den Zugriff auf alle QuickPascal-Befehle sichert er sich dabei durch den simplen Aufruf eines Befehls aus dem Optionen-Menü.

Der Editor

Wer von Turbo Pascal oder einer anderen Entwicklungsumgebung auf QuickPascal umsteigt, der wird mit der Bedienung des Editors unter Umständen etwas Schwierigkeiten haben, da die verschiedenen Editor-Befehle zum Einfügen, Löschen, Verschieben von Zeichen etc. über andere Tastenkombinationen aufgerufen werden, als er es bisher gewohnt war.

QuickPascal zwingt dem Anwender jedoch nicht seine Vorstellungen über die Bedienung des Editors auf. Vielmehr kann der Anwender über das beigefügte Programm QPMKKEY eigene Tastaturtabellen erstellen, mit deren Hilfe er die über 60 Funktionen des Editors auf beliebige Tastenkombinationen abbilden kann.

Und wem die Erstellung einer solchen Tabelle mit zu viel Arbeit verbunden ist, der kann auf mehrere vordefinierte Tabellen zurückgreifen, die im Lieferumfang von QuickPascal enthalten sind.

Einen ganz besonderen Leckerbissen stellt die Fähigkeit des Editors dar, Ihr Pascal-Programm bereits während der Eingabe zu analysieren und es dadurch in Befehle, Ausdrücke und Kommentare zu scheiden. Die nämlich können auf



Bild 2: Der Editor von QuickPascal markiert die verschiedenen Bestandteile eines Pascal-Programms auf Wunsch mit unterschiedlichen Farben bzw. Attributen.

Wunsch automatisch in unterschiedlichen Farben dargestellt werden, was der Lesbarkeit des Listings am Bildschirm sehr zu Gute kommt. Die Farben der einzelnen Komponenten können Sie übrigens selbst bestimmen und dadurch die Farbkombinationen wählen, die Ihnen am geeignetsten erscheinen.

Hilfe gibt's immer

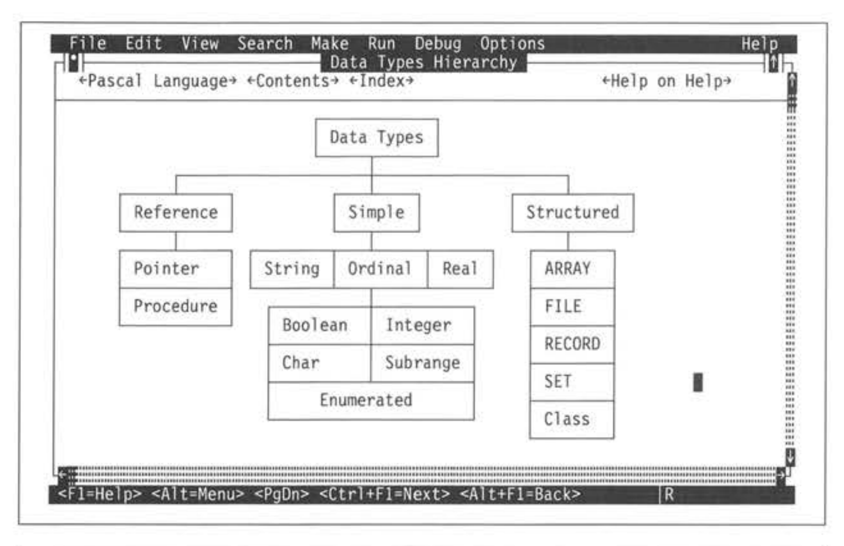
Ein weiteres Leistungsmerkmal von QuickPascal ist sein kontextsensitives Help-System, das sich bereits in anderen Quick-Compilern bewährt hat und Anfänger wie Profis gleichermaßen unterstützt. Der Begriff »kontextsensitiv« weist darauf hin, daß QuickPascal bei der Betätigung der Hilfe-Taste (F1) nicht irgendeinen Hilfetext auf den Bildschirm bringt, sondern individuell auf die Situation eingeht, in der sich der Anwender befindet.

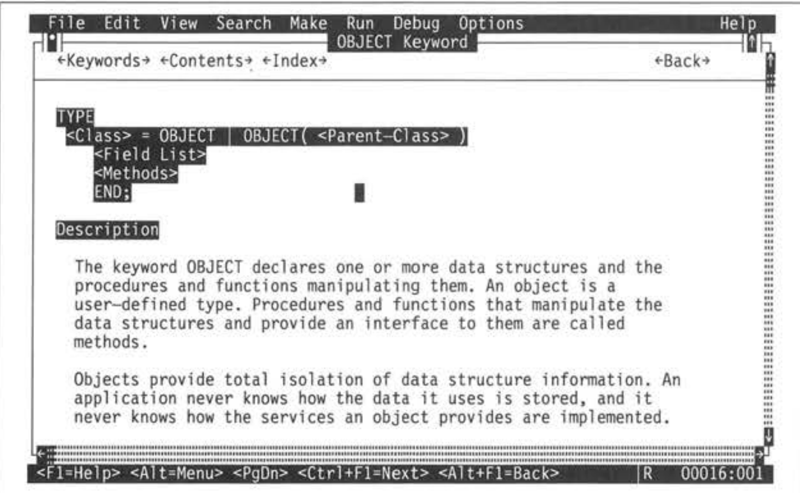
Innerhalb des Hauptmenüs fördert die Betätigung dieser Taste deshalb einen Hilfstext mit Informationen über das jeweils aktuelle Menü zu Tage, genau wie ein »Hilferuf« innerhalb einer Dialogbox zur Anzeige von Informationen über die Bedeutung dieser Dialogbox und ihrer verschiedenen Felder führt.

Darüber hinaus erlaubt das Hilfesystem von QuickPascal auch den Zugriff auf allgemeine Informationen mittels der sogenannten »Hyperlinks« – Verweise, die auf einen anderen Hilfetext zeigen und deren Anwahl zur Anzeige dieses Hilfetexts führt.

Auf diese Art und Weise hat der Anwender jederzeit Zugriff auf allgemeine Informationen (über die QuickPascal-Umgebung, die Sprache Pascal, ASCII-Tabelle etc.) sowie einen Index, in dem alle verfügbaren Stichwörter in alphabetischer Reihenfolge aufgeführt sind.

Neben diesen expliziten Hyperlinks kennt das Hilfesystem auch die impliziten Hyperlinks. Jedes Wort, das innerhalb des Help- oder Edit-Fenster mit Hilfe der Maus oder der Tastatur markiert wird, fungiert als ein solcher impliziter Hyperlink. Wird nach der Markierung dieses Begriffs





Bilder 3 und 4: Zwei Hilfeseiten, die mit Hilfe expliziter und impliziter Hyperlinks auf den Bildschirm geholt wurden.

die Hilfe-Taste betätigt, sucht QuickPascal nach einem Hilfetext, der Informationen im Zusammenhang mit dem markierten Begriff enthält und wird – dank des Umfangs der Help-Datei – dabei in der Regel fündig.

Insbesondere gilt dies auch für alle Pascal-Befehle, Operatoren und die Standard-Funktionen bzw. Prozeduren wie `writeln`, `gotoxy` etc., für die QuickPascal jeweils eine eigene Hilfeseite bereithält. In der wird dann analog zum Handbuch die genaue Syntax dieses Befehls, seine Aufgabe, die zugehörige Unit etc. beschrieben und auch ein kleines Demo-Programm angeboten, das die Verwendung dieses Befehls im Rahmen eines größeren Kontexts verdeutlicht.

Wie auch jeder andere Text innerhalb einer Hilfeseite, kann das Demo-Programm bzw. Teile daraus ohne weiteres vom Help- in das Edit-Fenster kopiert werden und Ihnen so Tipparbeit abnehmen.

Die *Bilder 3 und 4* verdeutlichen die Arbeit mit expliziten und impliziten Hyperlinks. Zunächst wurde über den Hilfe-Befehl im Hauptmenü eine Hilfe-Seite mit expliziten Hyperlinks auf einige allgemeine Themen angefordert und daraus die Hilfe-Seite mit Informationen über die Datentypen von QuickPascal ausgewählt (*Bild 3*). Innerhalb dieser Seite wurde der Maus-Cursor dann auf den Begriff `class` bewegt und dann der rechte Mausknopf niedergedrückt, um Informationen über diesen Begriff zu erhalten. Das Resultat sehen Sie im *Bild 4*.

Geht's noch schneller?

Diese Frage kommt unwillkürlich auf, wenn man die Kompilierungsgeschwindigkeit von QuickPascal betrachtet und fast hat man den Eindruck, QuickPascal würde jeden Pascal-Befehl einfach eins zu eins in einen Maschinensprache-Befehl übertragen, so schnell »fliegt« der Compiler durch den Quelltext. Dem ist natürlich nicht so, doch erreicht QuickPascal dank interner Optimierungen Kompilierungsgeschwindigkeiten, mit denen sich im Augenblick kein anderer Compiler – auch nicht Turbo Pascal – messen kann.

Dabei ist QuickPascal Source-kompatibel zu Turbo Pascal, so daß Turbo-Pascal-Programme ohne Veränderungen in die QuickPascal-Umgebung übernommen werden können. Erreicht wird dies durch Nachbildung der Turbo-Syntax, angefangen bei den Pascal-Befehlen bis hinunter zu den verschiedenen Compiler-Pragmas, mit deren Hilfe aus dem Quelltext heraus Einfluß auf die Kompilierung (Ein-/Ausschalten der Stack- und Typüberprüfungen etc.) genommen werden kann.

Ebenfalls kompatibel zu Turbo Pascal sind natürlich die Units, die eine wichtige Rolle innerhalb der Gesamtkonzeption spielen. Sie dienen der Erstellung von Programmodulen, die – wie das Hauptprogramm – kompiliert eine Größe von maximal 64 Kbyte nicht überschreiten dürfen und zu einem Programm beliebiger Größe kombiniert werden können.

Auch dienen sie zur Aufnahme der Standard-Prozeduren und -Funktionen und so wird QuickPascal mit fünf vordefinierten Quick-Units ausgeliefert, in denen sich alle Standard-Prozeduren und -Funktionen von Turbo Pascal und darüber hinaus sogar noch einige Prozeduren mehr finden.

Neben Routinen zur Ein-/Ausgabe, zur Speicherverwaltung und dem Zugriff auf die zahlreichen DOS- und BIOS-Funktionen wird auch eine umfangreiche Grafikbibliothek angeboten, die mit allen bekannten Video-Standards (MDA, Hercules, CGA, EGA, VGA und MCGA) zusammenarbeitet und geräteunabhängig konzipiert wurde.

Eine interessante Erweiterung gegenüber Turbo Pascal stellt z.B. die Möglichkeit dar, Programme speziell für die Ausführung auf einem 80286-Prozessor zu kompilieren, wodurch von den erweiterten Möglichkeiten dieses Prozessors Gebrauch gemacht werden kann, und das Programm schneller ausgeführt wird. Interessant sind aber auch andere Erweiterungen, wie z.B. die Einführung eines neuen Datentyps, des `CSTRING`, der eine Zeichenkette im C-Format auf-

nimmt und wie ein Pascal-String in Verbindung mit allen String-Prozeduren eingesetzt werden kann.

```
type KFZ = object
           X, Y,                                { augenblickliche Position }
           Richtung : integer;                       { Bewegungsrichtung }
           Tempo    : real;                 { aktuelle Geschwindigkeit }
           procedure Init( XPos, YPos : integer ); { Initialisierung }
           procedure Beschleunige;                    { KFZ beschleunigt }
           procedure Bremse;                                { KFZ bremst }
           procedure Lenke( r : integer );      { KFZ ändert Richtung }
         end;
```

◂◂ Bild 6: Eine mögliche Deklaration des Objekts Kraftfahrzeug unter QuickPascal

Der Kammerjäger ist immer zur Hand

Auch wenn Programme einmal nicht auf Anhieb so funktionieren, wie man sich das eigentlich vorgestellt hat (und das kommt bekanntlich öfters vor, als es einem lieb ist), läßt QuickPascal den Anwender nicht im Stich. Im Gegenteil, der integrierte Debugger unterstützt den Anwender bei der Fehlersuche, indem er ihm die Möglichkeit bietet, das Programm Befehl für Befehl auszuführen und am Bildschirm die jeweils ausgeführte Befehlszeile zu betrachten.

Dadurch wird es nicht nur möglich, den Programmverlauf bis ins kleinste Detail nachzuvollziehen, sondern nach der Ausführung eines Befehls kann auch angehalten werden, um beispielsweise den Inhalt einer Variablen zu betrachten oder zu verändern.

Dieser Aufgabe dient das Debug-Fenster, das über einen Befehl im `View`-Menü geöffnet werden kann und dann neben den verschiedenen Edit-Fenstern auf dem Bildschirm erscheint. Es dient der permanenten Überwachung von Variableninhalten, wobei der Anwender dazu lediglich den Namen der zu betrachtenden Variablen (bzw. einen beliebigen Ausdruck) am Anfang einer noch leeren Zeile eingeben und dann die Return-Taste betätigen muß.

Während er die Programmausführung verfolgt, wird der aktuelle Wert der Variablen immer in diesem Fenster angezeigt, so daß die Auswirkungen der verschiedenen Pascal-Befehle auf die einzelnen Variablen sofort deutlich werden. Dies gilt übrigens nicht nur für einfache Datentypen wie `INTEGER`, `STRING`, `CHAR` und `BOOLEAN`, sondern auch für komplexere Typen wie Zeiger, Arrays, Records etc.

Soll die Programmausführung nicht Schritt für Schritt nachvollzogen, sondern erst mit dem Erreichen einer bestimmten Programmzeile gestoppt werden, ist ebenfalls auf QuickPascal Verlaß; es erlaubt die Definition mehrerer Breakpoints, bei deren Erreichen die Programmausführung gestoppt und die aktuelle Programmzeile im Edit-Fenster angezeigt wird, so daß man erkennen kann, welcher Breakpoint durchlaufen wurde.

Darüber hinaus können Breakpoints auch mit einer beliebigen Bedingung in Form eines Pascal-Ausdrucks verknüpft werden, der ein `BOOLEAN`-Resultat im Sinne von `TRUE` oder `FALSE` liefern muß. Bei jedem Durchschreiten des Breakpoints wertet QuickPascal diese Bedingung aus und hält die Programmausführung erst dann an, wenn die Auswertung TRUE ergibt.

```
 File  Edit  View  Search  Make  Run  Debug  Options                        Help
                    [1] C:\QP\SAMPLES\SORTDEMO.PAS

    TextColor( menu_status );
    TextBackground( menu_back );
    GotoXY( menu_left + 21, Ord(current_sort) + menu_top + 3 );
    Write( ((finish_time - start_time) / ticks_per_sec ):7:2 );

    IF sound_on THEN
        BEGIN
        Sound( current_row * scale_frequency );
 ------------------------------- DEBUG ------------------------------------
finish_time : 1280591
(finish_time-start_time) / ticks_per_sec : 0.0
sort : (8170:064A,8170:02B8,8170:0442,8170:036E,8170:0918,8170:0738)
sort_array : ((7,8),(10,11),(8,9),(13,14),(43,14),(14,15),(15,1),(4,5),(33,4),
Ord(current_sort) + menu_top + 3 : 9

<F1=Help> <Alt=Menu> <F5=Run> <F6=Window> <F8=Trace> <F10=Step>     00131:001
```

Bild 5: Mit Hilfe des Debug-Fensters kann der Inhalt von Variablen während der Programmausführung überwacht werden.

Praktisch sind diese »konditionalen« Breakpoints z.B. in Verbindung mit `FOR`-Schleifen, wenn man die Progammausführung erst nach dem n-ten Durchlauf der Schleife anhalten möchte. Als Abbruchbedingung würde man dann definieren: `Schleifenvariable = n`

Besonders tückische Fehler, wie etwa das fehlerhafte Laden einer Variablen an einer unbekannten Programmstelle lassen sich damit allerdings nicht aufdecken, doch haben sich die QuickPascal-Entwickler auch für eine solche Situation einen Trick einfallen lassen. Im Mittelpunkt steht dabei wiederum ein Ausdruck, der als Resultat TRUE oder FALSE liefern muß und beim Ergebnis TRUE zum Abbruch der Programmausführung führt.

Im Gegensatz zu einem konditionalen Breakpoint wird dieser Ausdruck jedoch nicht erst beim Durchlaufen des Breakpoints, sondern nach der Ausführung *jedes* Befehls ausgewertet. Zwar verlangsamt dies die Programmausführung ganz beträchtlich, doch wird das Warten belohnt, wenn der Debugger nach Eintritt der Bedingung die Programmzeile anzeigt, die die fehlerhafte Anweisung enthält.

Alle reden von OOP, QuickPascal hat sie

Objektorientiertes Programmieren, derzeit in aller Munde, muß dank QuickPascal keine theoretische Angelegenheit mehr bleiben, denn in QuickPascal findet der Anwender die Grundkonzepte objektorientierten Programmierens verwirklicht, ohne dabei auf traditionelle Strukturen verzichten zu müssen.

▸▸ *Bild 7: Die Definition der Methode Init für die Objektklasse KFZ.*

```
procedure KFZ.Init( XPos, YPos : integer );

begin
  self.X := XPos;           { aktuelle Position merken }
  self.Y := YPos;
  self.Tempo := 0.0;          { KFZ bewegt sich nicht }
  self.Richtung := 1;       { Richtung initialisieren }
end;
```

QuickPascal greift auf zahlreiche Ideen zurück, die der Däne Bjarne Stroustrup in der von ihm entwickelten Computersprache C++ realisiert hat. QuickPascal gehört damit wie C++ zu den sogenannten hybriden Sprachen, die im Gegensatz zu den rein objektorientierten Systemen wie Smalltalk, Actor oder Eiffel Source-kompatibel zum bisherigen Stand der Software-Technik, den prozeduralen Sprachen, sind. OOPS-Puristen mögen nun anmerken, daß sich dem Programmierer dadurch längst nicht alle Möglichkeiten objektorientierten Programmierens erschließen, doch ist dem entgegenzuhalten, daß die von QuickPascal gebotenen Möglichkeiten auf absehbare Zeit voll und ganz ausreichen und es sicher eine ganze Weile dauern wird, bis die hier realisierten Ideen die gesamte Programmierer-Welt durchdrungen haben werden. Außerdem sollte man nicht vergessen, daß man beim Umstieg auf rein objektorientierte Systeme allen bisher entwickelten Programmcode gleich dem Papierkorb zuführen kann, weil diese Systeme mit nichts bisher bekanntem vergleichbar sind.

Was aber sind die Ideen, die OOPS von den Konzepten prozeduraler Sprachen abhebt?

Im Mittelpunkt steht das Objekt

Ausgangspunkt der OOP-Forschung, die bereits Ende der sechziger Jahre ihren Anfang nahm, sind einige – beinahe philosophische – Einsichten über das, was wir »Programmieren« nennen. Geht man der Natur dieser Tätigkeit auf den Grund, so kommt man schnell zu der Einsicht, daß bei der Programmierung eines Computers immer der Versuch unternommen wird, die Wechselwirkung zwischen den Teilen eines Systems in der Welt eines Computers abzubilden. Und je komplexer diese Wechselwirkungen sind, desto abstrakter muß auch die Sprache sein, derer wir uns zu ihrer Beschreibung bedienen.

Unter diesem Gesichtspunkt hat die Informatik in den letzten drei Jahrzehnten auf ihrem Weg von der Maschinen- über die Assemblersprache bis hin zu Basic, Pascal und C schon große Fortschritte erzielt. Trotzdem gibt es Problemstellungen, die wir zwar mit unserem Denken erfassen und begreifen, sie aber nur schwer mit Hilfe prozeduraler Sprachen ausdrücken können. Wir kennen dieses Phänomen aus dem täglichen Leben, denn auch wir haben es manchmal schwer, Dinge zu verbalisieren, die wir gedanklich gut erfassen können, ganz einfach, weil uns die »Worte« fehlen.

Die objektorientierte Programmierung versucht nun, unseren Diskurs mit dem Computer auf eine höhere Abstraktionsebene zu heben, die Computer-»Sprache« gleichsam mehr den Kategorien unseres Denkens anzugleichen. Man spricht hier auch von einem neuen »Paradigma«, eine neue Art, die Dinge zu sehen und in formalisierte Beschreibungen umzusetzen.

Wir – das hat uns die Wahrnehmungsphysiologie gelehrt – erfassen unsere Umwelt als eine Ansammlung von Objekten, denen wir Attribute und Fähigkeiten zuordnen und die wir in Bezug zu anderen Objekten setzen. Diesem Gedanken folgend, stellt die objektorientierte Programmiertechnik das Objekt und seine Beschreibung in den Mittelpunkt eines Programms. Wie auch einem Objekt seine Attribute (Größe, Farbe, Form, Inhalt etc.) innewohnen, so nimmt auch das QuickPascal-Objekt seine Attribute in Form von Variablen auf, deren Manipulation ihm allein vorbehalten bleibt.

Da aber auch die Fähigkeiten eines Objekts etwas dem Objekt innewohnendes sind und ihm nicht etwa von außen herangetragen werden, schließt eine Objektdefinition auch die Fähigkeiten des Objekts ein, die in Form sogenannter Methoden durch den Programmcode verkörpert werden. Die Verbindung von Daten und Programmcode bezeichnet man als Kapselung (Encapsulation). Sie stellt eines der drei elementaren Prinzipien objektorientierten Programmierens dar. Die Definition des Objekts Kraftfahrzeug innerhalb eines Verkehrssimulationsprogramms könnte unter QuickPascal z.B. so aussehen wie in *Bild 6*.

Wie Bild 6 zeigt, erfolgt die Deklaration eines Objekts grundsätzlich in Verbindung mit dem `TYPE`-Befehl durch Angabe des Schlüsselworts `OBJECT`. Darauf folgen die Attribute des Objekts in Form von Variablendeklarationen, ähnlich denen innerhalb eines `RECORDS`. An die Deklaration der Attribute schließen sich die Methoden-Deklarationen an, die den `FORWARD`-Deklarationen von Prozeduren und Funktionen gleichen. Tatsächlich werden die Methoden eines Objekts wie ganz normale Prozeduren und Funktionen behandelt, denen Parameter übergeben werden und die, im Falle einer Funktion, selbst ein Resultat an den Aufrufer zurückliefern können.

Der Deklaration der Methoden folgt ihre Definition innerhalb des Programmlistings, wobei dem Namen der Methode der Name des Objekts gefolgt von einem Punkt vorangehen muß. Die Initialisierungsprozedur der Objektklasse KFZ könnte z.B. so wie in *Bild 7* aussehen. Beachten Sie bitte, daß der Prozedur zwar die beiden Parameter `XPos` und `YPos`, nicht aber eine Information über das zu initialisierende Objekt übergeben wird. Die nämlich geht aus dem Aufruf der Methode hervor, wie ihn das *Bild 8* zeigt.

```
begin                                   { beliebige Prozedur }

var Auto : KFZ; { Auto ist ein Instanz der Objektklasse KFZ }

begin
  New( Auto );                          { Auto erzeugen }
  Auto.Init( 15, 30 );                  { Auto initialisieren }
end;
```

◂◂ *Bild 8: So kann eine Instanz der Objektklasse KFZ erzeugt und mit Hilfe der Methoden der Objektklasse manipuliert werden.*

In Bild 8 wird zunächst mit Hilfe des VAR-Befehls ein Objekt vom Typ KFZ definiert. Dadurch wird eine sogenannte »Instanz« der Objektklasse KFZ erzeugt. Auf Instanzen kann jedoch erst zugegriffen werden, nachdem für sie mit Hilfe der New-Prozedur Speicher allokiert wurde. Dies hängt damit zusammen, daß die VAR-Deklarationen nicht die Instanz selber, sondern grundsätzlich einen Zeiger auf die Instanz erzeugen, der mittels New mit der eigentlichen Instanz verbunden werden muß. (Warum das so ist, wird in wenigen Augenblicken deutlich werden.)

Natürlich kann auf diese Art und Weise nicht nur eine Instanz, sondern annähernd beliebig viele Instanzen erzeugt und mit Hilfe der verschiedenen Methoden der Objektklasse manipuliert werden. Nur die Grenze des Speichers schränkt die Anzahl der gleichzeitig existierenden Instanzen ein, doch wird für jede Instanz nur der Speicher für die Objektvariablen (und einige zusätzliche Verwaltungs-Bytes) benötigt, da die Methoden nur einmal kodiert und dann von allen Instanzen einer Objektklasse gemeinsam genutzt werden.

Nachdem eine Instanz mittels New erzeugt wurde, kann sie mit Hilfe der Methoden der Objektklasse, zu der sie gehört, manipuliert werden. Um eine solche Methode aufzurufen, muß zunächst der Name der Instanz, dann ein Punkt und dann der Name der Methode angegeben werden. Dahinter können, wie bei einem normalen Prozedur- oder Funktionsaufruf, die zu übergebenen Argumente in Klammern folgen.

Der Terminus »Aufruf« trifft die Dinge dabei aber nur aus der Sicht des prozeduralen Paradigmas. Ein OOPS-Programmierer würde eine Anweisung wie

```
Auto.Init( 15, 30 );                    { Auto initialisieren }
```

wohl eher als Auftrag bezeichnen und sie wie folgt kommentieren:

»Sende an das Objekt Auto die Nachricht, daß es sich mit den Parametern 15 und 30 initialisieren soll«.

Hier wird dem Objekt also eine aktive Rolle zugewiesen. Nicht länger modifzieren Prozeduren und Funktionen ihnen übergebene, passive, Daten, sondern Objekte werden durch den Empfang von Nachrichten (Messages) dazu veranlaßt, sich selbst zu manipulieren. Tatsächlich stellt ein Programm aus der Sicht eines OOPS-Programmierers eine Art großer Bühne dar, auf der die einzelnen Instanzen als die verschiedenen Schauspieler agieren, denen der Regisseur (das Hauptprogamm) Anweisungen in Form von Nachrichten übermittelt und sie dadurch zum Handeln veranlaßt.

Dabei kommt ein weiteres wichtiges Prinzip der objektorientierten Programmierung zum tragen, das Prinzip der Datenabstraktion. Nur für das Objekt selbst sind nämlich die Objektvariablen, also seine Attribute, sichtbar und nur es selbst darf auf diese Attribute zugreifen. Jedem anderen Objekt und jeder normalen Prozedur oder Funktion ist der Zugriff auf das Innerste eines Objekts versperrt. Wie der Zugriff auf diese Attribute erfolgt zeigt Bild 7, in dem die Definition der Init-Methode des KFZ-Objekts aufgeführt ist.

Das kurze Listing zeigt, daß der Zugriff auf die Attribute mit Hilfe des self-Parameters erfolgt, der jeder Methode als implizites Argument übergeben wird und den Zugriff auf die Instanz ermöglicht, an die die entsprechende Nachricht gesandt wurde. Anstatt vor jeder der vier Zuweisungen hier self. zu schreiben, könnte man sich auch des with-Befehls bedienen, die Methode also durch den Befehl with self do einleiten.

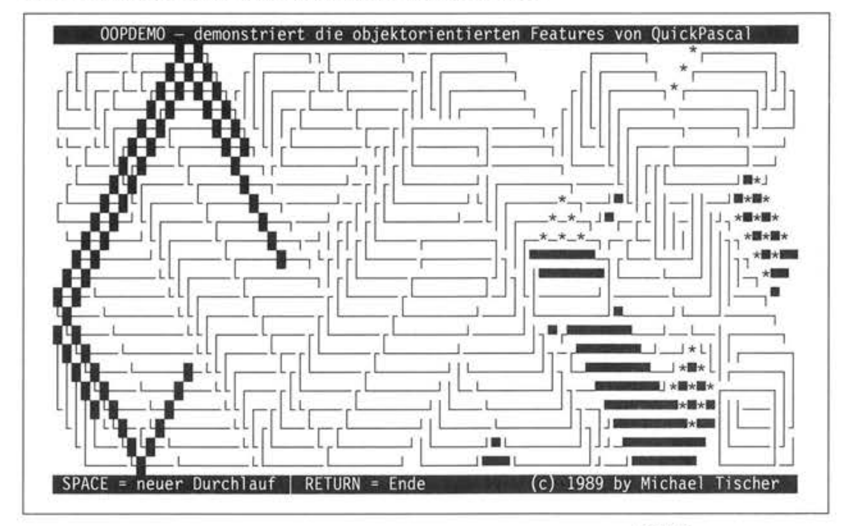


Bild 9: Das Demo-Programm OOPDEMO läßt verschiedene geometrische Objekte über den Bildschirm tanzen.

In der Klassengesellschaft haben Stammbäume wieder Gewicht

Zwei weitere wichtige Säulen des OOPS-Konzepts, das Prinzip der Vererbung (Inheritance) und des Polymorphismus (Polymorphism) möchte ich Ihnen am Beispiel des Demo-Programms deutlich machen, dessen Listing Sie auf den folgenden Seiten finden. *Bild 9* läßt bereits erahnen, worum es in diesem Programm geht.

Mit Hilfe der objektorientieren Erweiterungen von QuickPascal werden hier die drei geometrischen Objekte Punkt, Rechteck und Dreieck definiert und Methoden bereitgestellt, die die Objekte dazu bringen, auf dem Bildschirm zu erscheinen und sich in einer bestimmten Richtung über den Bildschirm zu bewegen, wobei sie an den Bildschirmrändern reflektiert werden und

dann eine neue Bewegungsrichtung einschlagen.

Zunächst wird dazu die Objektklasse Punkt definiert. Innerhalb jeder Instanz dieser Klasse wird die Position der Instanz als Bildschirmkoordinate in den Variablen X und Y sowie die Bewegungsrichtung in der Variablen Richtung gespeichert. Da es vier Bewegungsrichtungen gibt, die jeweils in diagonaler Richtung auf eine der vier Bildschirmecken zeigen, wird die Bewegungsrichtung als Wert zwischen 1 und vier festgehalten und als Index in das globale Array BewDa betrachtet. In diesem Array werden für die vier Bewegungsrichtungen die Offsets in X- und Y-Richtung sowie die möglichen Bewegungsrichtungen für den Fall eines Zusammenpralls mit dem Bildschirmrand festgehalten.

Punkt verfügt über sechs Methoden, die teils als Prozeduren, teils als Funktionen realisiert sind:

• pinit initialisiert die Instanz, trägt ihre Bildschirmposition in die Variablen X und Y ein und wählt eine zufällige Bewegungsrichtung aus, die es in die Variable Richtung einträgt.
• draw zeichnet die Instanz an der Bildschirmposition, an der sie sich gerade befindet.
• move verschiebt die Instanz in ihrer augenblicklichen Bewegungsrichtung über den Bildschirm und wählt eine neue Richtung, falls die Instanz mit dem Bildschirmrand kollidiert.
• GetX liefert die Bildschirmspalte, in der sich die Instanz befindet.
• GetY liefert die Bildschirmzeile, in der sich die Instanz befindet.
• check überprüft, ob die Instanz, stellt man sie ab der übergebenen Bildschirmposition dar, noch auf den Bildschirm paßt oder mit dem Bildschirmrand kollidiert.

Bevor unser Augenmerk der Realisation dieser Methoden gelten soll, möchte ich Ihren Blick auf die Deklaration der Objekte rechteck und dreieck leiten, die innerhalb des Listings der Deklaration von punkt folgen. Das Besondere an ihnen ist die Angabe des Objektnamens punkt in Klammern hinter dem Schlüsselwort object. Diese Art der Deklaration weist QuickPascal nämlich darauf hin, daß rechteck und dreieck Nachfahren der Objektklasse punkt sind. Dadurch »erben« sie alle Methoden und Variablen ihres Vorfahrens, können neben den eigenen Methoden und Variablen auch auf sie zugreifen.

Hinter dem Prinzip der Vererbung verbirgt sich die Idee, immer komplexere Objekte aus einfacheren Objekten zusammenzusetzen, so wie auch in unserer Umwelt viele Objekte (etwa PCs) aus in sich abgeschlossenen Objekten bestehen, deren Verbindung miteinander erst das Ganze ergibt. So könnte man aus rechteck und dreieck wiederum beliebig viele Objektklassen hervorgehen lassen, die dann neben den Methoden und Variablen von punkt, auch die Merkmale ihrer direkten Vorfahren, also rechteck oder dreieck, erben.

Während rein objektorientierte Systeme auch die »multiple inheritence« erlauben, die Erstellung eines Objekts aus mehreren Vorfahren, ist dies bei hybriden Sprachen in der Regel nicht möglich. Auch QuickPascal sieht nur die Vererbung von Merkmalen durch jeweils einen Vorfahren vor, doch zeigt die praktische Arbeit, das dies den Programmierer nur in solchen Fällen einengt, in denen bereits andere Gründe den Einsatz reiner OOPS-Systeme sinnvoll erscheinen lassen.

Fahren wir mit der Betrachtung der Objektklasse rechteck fort. Auch Rechtecke besitzen eine (Ursprungs-)Koordinate in Bezug auf den Bildschirm und eine Bewegungsrichtung, doch müssen diese Informationen innerhalb der Deklaration nicht nochmals aufgeführt werden, da sie von punkt übernommen werden und dadurch (unsichtbar) bereits vorhanden sind. Hinzu kommen jedoch die Variablen XLen und YLen, die die Länge der horizontalen und vertikalen Seiten des Rechtecks widerspiegeln und dadurch aus einem Punkt ein Rechteck formen.

Beachten Sie bitte, daß diese Variablen und auch die zusätzlich definierten Methoden nur der Objektklasse rechteck und deren Nachfolgern, nicht aber der Objektklasse punkt zur Verfügung stehen. Die Vererbung wirkt sich also immer nur in Richtung auf die Nachfahren aus. Die Vorfahren bleiben grundsätzlich unbeeinflußt!

Da zur Initialisierung eines Rechtecks nicht nur dessen Bildschirmposition, sondern auch die Länge der beiden Seiten bekannt sein muß, wird die Objektklasse rechteck mit der Methode rinit versehen, die diese Parameter erfaßt und in die Variablen der Instanz einträgt. Darüber hinaus wird rechteck mit zwei eigenen Versionen der Methoden draw und check versehen, denn durch die unterschiedliche Form, kann rechteck nicht auf die gleichnamigen Methoden von punkt zurückgreifen. Da rechteck jedoch bereits zwei Methoden mit diesen Namen von seinem Vorgänger geerbt hat, müssen die beiden Methoden-Deklarationen mit dem Schlüsselwort override versehen werden, damit QuickPascal erkennt, daß rechteck über eigene draw- und check-Methoden verfügt und nicht die geerbten Methoden zum Einsatz kommen sollen.

Zwar könnte man dieses Problem leicht umgehen, indem man der draw- und check-Methode von rechteck einfach einen anderen Namen verleiht, doch werden wir gleich noch sehen, daß die hier eingeschlagene Vorgehensweise in Bezug auf das Konzept des Polymorphismus besonders wichtig ist.

Die Deklaration der Objektklasse dreieck unterscheidet sich nur wenig von der rechteck-Deklaration. Auch sie ergänzt die von punkt übernommenen Variablen um eine weitere Information, die die Länge der Grundseite des Dreiecks aufnimmt. Da es sich hier grundsätzlich um gleichschenklige Dreiecke handelt, kann die

Länge der beiden Schenkel aus der Länge der Grundseite berechnet werden.

Mit der Methode dinit verfügt auch dreieck über eine eigene Initialisierungsmethode und durch Überschreiben der von punkt übernommenen check- und draw-Methoden geht es auf seine spezielle geometrische Form ein.

Die Methoden im Beispielprogramm OOPDEMO

Bis auf die Methode move sind alle Methoden von punkt recht trivial und einfach zu verstehen. Pinit lädt die übergebenen Parameter in die Instanzvariablen, check prüft, ob sich die Instanz im sichtbaren Bildschirmbereich befindet, GetX und GetY liefern die Koordinate der Instanz zurück und draw setzt den Cursor mittels der Standardprozedur GotoXy auf die Bildschirmposition des Punktes und gibt dann durch einen Aufruf von Write das Zeichen mit dem ASCII-Code 219 aus, das auf dem Bildschirm als inverser Zeichenblock erscheint.

Wesentlich komplexer und interessanter ist da schon die move-Methode. Sie entfernt die übergebene Instanz zunächst vom Bildschirm, indem sie die Textfarbe für Ausgaben über Write und Writeln auf 0 setzt und dann die draw-Methode der Instanz aufruft. Da die das Objekt eben genau mit Hilfe der Write-Prozedur auf dem Bildschirm aufbaut und die Textfarbe nun die Farbe des Bildschirmhintergrundes angenommen hat, entfernt sie die Instanz ungewollt vom Bildschirm.

Danach testet move mit Hilfe der check-Methode, ob sich die Instanz an der neuen Position, die sie nach ihrer Verschiebung inne haben soll, noch innerhalb des sichtbaren Bildschirmbereichs befindet. Ist dem nicht so, sucht move eine Bewegungsrichtung, in der die Instanz verschoben werden kann, wählt diese als neue Bewegungsrichtung aus und paßt die Bildschirmkoordinaten der Instanz entsprechend an.

An die neue Bildschirmposition bringt move die Instanz, indem es wieder eine sichtbare Textfarbe einstellt und dann erneut eine Nachricht an die draw-Methode der Instanz sendet.

Das alles macht move natürlich für Instanzen aus der Objektklasse punkt, doch der Clou ist, daß ein Programm auch auf diese Methode zurückgreifen kann, um die Nachfahren von punkt, also rechteck und dreieck über den Bildschirm zu verschieben, ohne daß sie die move-Methode durch eine für sie angepaßte Version ersetzen.

Möglich wird dies durch Vererbung und Polymorphismus. Ersteres trägt dazu bei, daß die Nachfahren von punkt auch über die move-Methode verfügen, sie mittels einer Nachricht also aktivieren können. Dem Polymorphismus ist es andererseits zu verdanken, daß move nicht grundsätzlich die Methoden draw und check aus der Objektklasse punkt anspricht, sondern jeweils eine Nachricht an die draw- und check-Methode der Objektklasse sendet, zu der die Instanz gehört, die move aktiviert hat und auf die sich der self-Parameter bezieht.

Dieser auf den ersten Blick recht einfach wirkende Mechanismus hat auf die Codegenerierung ganz entscheidenden Einfluß, denn QuickPascal weiß bei der Kodierung der move-Methode noch nicht, welche draw- und check-Methode es (im prozeduralen Sinne) aufrufen muß. Die aus der Objektklasse punkt, oder die aus rechteck oder gar der Objektklasse dreieck?

Diese Frage kann zum Zeitpunkt der Kompilierung einfach noch nicht beantwortet werden, und so ist QuickPascal gezwungen, ein sogenanntes »late binding« durchzuführen, das nicht bereits zum Zeitpunkt der Kompilierung, sondern erst unmittelbar während der Programmausführung festlegt, an welche der verschiedenen draw- und check-Methoden eine Nachricht gesandt werden muß. Realisiert wird dies, indem für jede Objektklasse eine interne Tabelle mit den Adressen der zugehörigen Methoden angelegt und eine Objektinstanz über den New-Befehl mit dieser Tabelle verbunden wird. Bei einem Aufruf einer Methode wird daher zunächst die Objektklasse ermittelt, zu der die bearbeitete Instanz gehört, und aus der Tabelle dieser Instanz dann die Adresse der aufzurufenden Methode geladen.

Vergißt der Programmierer, eine Instanz über den New-Befehl mit dieser Tabelle zu verbinden, zeigt der zugehörige Tabellen-Zeiger in die Wüste und genau dorthin wird auch QuickPascal beim ersten Senden einer Nachricht an eine Methode springen. Um dies zu verhindern, empfiehlt es sich, während der Programmentwicklung den Compilerschalter $M einzuschalten, der eine Instanz-Überprüfung vor dem Absenden jeder Nachricht an eine Methode durchführt. Stellt er dabei fest, daß die Verbindung zur Instanz-Tabelle noch nicht aufgebaut wurde, bricht er die Programmausführung mit einem Laufzeit-Fehler ab.

Mit diesem Wissen gewappnet, wird es ihnen nicht schwerfallen, die draw- und check-Methoden von rechteck und dreieck zu verstehen. Etwas Neues halten dabei allein die beiden check-Methoden bereit. Sie nämlich bedienen sich der check-Methode aus der Objektklasse punkt, um festzustellen, ob jeweils alle Ecken der Figur noch auf den Bildschirm passen. Nur wenn dies gewährleistet ist, kann die Instanz von move an die angegebene Bildschirmposition verschoben werden. Da die beiden Methoden den gleichen Namen wie punkt.check tragen, müssen sie sich des Befehlswortes inherited bedienen, damit der Compiler erkennt, das hier nicht eine (rekursive) Nachricht an sich selbst gesendet werden soll, sondern die Nachricht vielmehr für die geerbte check-Methode bestimmt ist.

▸▸ *Listing 1: OOPDEMO.PAS*

Das Hauptprogramm

Innerhalb des Hauptprogramms werden auf der Basis des Zufallsgenerators so viele Objekte erzeugt, wie die Konstante `ANZ_OBJEKTE` angibt. Um diese Objekte verwalten zu können, werden drei Arrays von dieser Größe angelegt. Eines für Punkte, eines für Rechtecke und eines für Dreiecke. Leider können die verschiedenen Objekte nicht in einem Array verwaltet werden, denn wie es nicht möglich ist, verschiedenartige `records` in einem Array unterzubringen, so können auch Instanzen aus unterschiedlichen Objektklassen nicht in einem Array zusammengefaßt werden.

Nach der Erzeugung der Objekte tritt das Programm in eine Schleife, die erst beendet wird, wenn der Anwender die Leertaste oder Return betätigt. Innerhalb dieser Schleife werden die verschiedenen Objekte fortwährend mit Hilfe der `move`-Methode über den Bildschirm bewegt, was zumindest für meinen Geschmack ganz witzig aussieht.

Perspektiven

Obwohl der Grundstein zur objektorientierten Programmierung bereits im Jahre 1967 von den Forschern Nygaard und Dahl mit ihrer Sprache »Simula« gelegt wurde, breiten sich die damit verbundenen Ideen erst langsam in der Programmierer-Gemeinschaft aus, und es wird wohl noch eine ganze Weile dauern, bis das objektorientierte Paradigma unseren heutigen Programmierstil ablösen wird. Bereits jetzt aber wird deutlich, daß die objektorientierte Programmiertechnik zum Wohle der Programmierer eingesetzt werden kann, um große Software-Projekte durchschaubarer zu machen, die Wiederverwendbarkeit des Programmcodes zu steigern und die Fehleranfälligkeit zu reduzieren. Dem Programmierer eröffnen sich damit Perspektiven, die ihm die Möglichkeit bieten, auch in den 90er Jahren mit den wachsenden Anforderungen des Marktes Schritt halten zu können. QuickPascal ist dabei der erste und vielleicht entscheidende Schritt.

Michael Tischer

```
{*****************************************************************************
*  OOPDEMO : demonstriert die objektorientierten Features von QuickPascal    *
*            anhand mehrerer Objekte, die sich über den Bildschirm bewegen   *
*            und dabei vom Bildschirmrand reflektiert werden.                *
**__________________________________________________________________________**
*  Autor            : MICHAEL TISCHER                                        *
*  entwickelt am    : 22.07.1989                                             *
*  letztes Update am: 22.07.1989                                             *
*****************************************************************************}

{$M-}                          { Objekt-Initialisierung nicht mehr überprüfen }

program oopdemo;

uses Crt;                                      { Unit mit GotoXY, ClrEol etc. }

const ANZ_OBJEKTE = 10;                       { Anzahl zu erzeugender Objekte }

{== Objekt- und Typdeklarationen ============================================}

type RDaten = record                    { Daten für die Reflektion an einer Wand }
                NeueRichtung : array [1..3] of byte;          { Richtungscodes }
                MoveX, MoveY : integer;                     { Richtungsoffsets }
              end;

type punkt = object                            { das Basisobjekt ist der Punkt }
       X, Y     : integer;                      { Position auf dem Bildschirm }
       Richtung : byte;                                  { Bewegungsrichtung }
       procedure pinit( XPos, YPos : integer);
       procedure draw;                                         { Objekt malen }
       procedure move;                                   { Objekt verschieben }
       function  GetX : integer;                                { X-Pos holen }
       function  GetY : integer;                                { Y-Pos holen }
       function  check( NeuX, NeuY : integer ) : boolean;
     end;

type rechteck = object( punkt )                { Rechteck folgt aus dem Punkt }
       XLen, YLen : integer;                                  { Seitenlängen }
       procedure rinit( XPos, YPos, dX, dY : integer );
       function check( NeuX, NeuY : integer ) : boolean; override;
       procedure draw; override;
     end;

type dreieck = object( punkt )               { ein gleichschenkliges Dreieck }
       SLen : integer;                                        { Seitenlängen }
       procedure dinit( XPos, YPos, dS : integer );
       function check( NeuX, NeuY : integer ) : boolean; override;
       procedure draw; override;
     end;

{== initialisierte globale Variablen (typisierte Konstanten) ================}

const BewDa : array [1..4] of RDaten =       { Bewegungsdaten für Reflektion }
 (
  ( NeueRichtung :( 4, 2, 3 ) ; MoveX :  1; MoveY : -1 ),
  ( NeueRichtung :( 3, 1, 4 ) ; MoveX : -1; MoveY : -1 ),
  ( NeueRichtung :( 2, 4, 1 ) ; MoveX : -1; MoveY :  1 ),
  ( NeueRichtung :( 3, 1, 2 ) ; MoveX :  1; MoveY :  1 )
 );

{== Methoden des Objekts Punkt ==============================================}
{*****************************************************************************
*  PUNKT.PINIT: lädt die Koordinate eines Punktes (bzw. eines nachfolgenden  *
*               Objekts) in die Objekt-Variablen X und Y und wählt eine zu-  *
*               fällige Bewegungsrichtung aus.                               *
**__________________________________________________________________________**
*  Eingabe : X, Y = Koordinate des Objekts                                   *
*  Ausgabe : keine                                                           *
*****************************************************************************}

procedure punkt.pinit( XPos, YPos : integer);
begin
  self.X := XPos; self.Y := YPos;                           { Position merken }
  self.Richtung := Random( 3 ) + 1;               { Bewegungsrichtung auslosen }
end;

{*****************************************************************************
*  PUNKT.CHECK : stellt fest, ob die Koordinate des übergebenen Bildschirm-  *
*                punktes noch innerhalb des Bewegungsbereichs der Objekte    *
*                auf dem Bildschirm liegt.                                   *
**__________________________________________________________________________**
*  Eingabe : X, Y = Bildschrimposition                                       *
*  Ausgabe : TRUE, wenn der Punkt noch innerhalb des Bewegungsbereichs liegt,*
*            sonst FALSE.                                                    *
*****************************************************************************}

function punkt.check( NeuX, NeuY : integer ) : boolean;

begin
  check := (NeuX >= 1) and (NeuX <= 80) and (NeuY >= 2) and (NeuY <=24);
end;

{*****************************************************************************
*  PUNKT.GETX : liefert die Spalte, in der sich ein Objekt vom Typ PUNKT     *
*               oder eines der nachfolgenden Objekte befindet.               *
**__________________________________________________________________________**
*  Eingabe : keine                                                           *
*  Ausgabe : Spalte des Objekts                                              *
*****************************************************************************}

function punkt.GetX : integer;

begin
  GetX := self.X;                          { Spaltenposition aus dem Objekt holen }
end;

{*****************************************************************************
*  PUNKT.GETY : liefert die Zeile, in der sich ein Objekt vom Typ PUNKT oder *
*               eines der nachfolgenden Objekte befindet.                    *
**__________________________________________________________________________**
*  Eingabe : keine                                                           *
*  Ausgabe : Zeile des Objekts                                               *
*****************************************************************************}
```

Listing 1: Fortsetzung und Ende.

```
function punkt.GetY : integer;
begin
  GetY := self.Y;                       { Zeilenposition aus dem Objekt holen }
end;

{****************************************************************************
*  PUNKT.DRAW : Malt ein Objekt vom Typ PUNKT in der aktuelle eingestellten  *
*               Ausgbefarbe                                                  *
**__________________________________________________________________________**
*  Eingabe : keine                                                           *
*  Ausgabe : keine                                                           *
*  Info    : Diese Methode wird von der Methode MOVE aufgerufen, um das      *
*            Objekt auf dem Bildschirm zu zeichnen oder vom Bildschirm zu    *
*            entfernen.                                                      *
****************************************************************************}
procedure punkt.draw;
begin
  GotoXY( self.X, self.Y );               { Cursor auf Objekt-Position setzen }
  write( '█' );                                            { Punkt zeichnen }
end;

{****************************************************************************
*  PUNKT.MOVE : verschiebt ein OBJEKT vom Typ PUNKT oder eines der nachfol-  *
*               genden Typen in der aktuellen Bewegungsrichtung über den     *
*               Bildschirm. Stößt das Objekt dabei an den Rand des Bewegungs-*
*               bereichs, wird die Bewegungsrichtung so verändert, daß das   *
*               Objekt sich weiter bewegen kann.                             *
*___________________________________________________________________________**
*  Eingabe : keine                                                           *
*  Ausgabe : keine                                                           *
****************************************************************************}
procedure punkt.move;

var i, r : byte;     { dienen der Berechnung einer neuen Bewegungsrichtung  }

begin
  with self do                          { auf die Felder des Objekts zugreifen }
    begin
      TextColor( 0 );                            { Zeichen unsichtbar machen }
      draw;                                              { Objekt ausblenden }
      if ( not check( X + BewDa[self.Richtung].MoveX,                { Crash? }
                      Y + BewDa[Richtung].MoveY ) ) then
        begin                                     { Ja, neue Richtung suchen }
          i := 0;
          repeat                               { nach einer neuen Richtung suchen }
            inc( i );                { Index für Richtungszähler inkrementieren }
            r := BewDa[Richtung].NeueRichtung[i];              { neue Richtung }
          until check( X + BewDa[r].MoveX, Y + BewDa[r].MoveY );
          Richtung := r;                      { neue Richtung gefunden, merken }
        end;
      inc( X, BewDa[Richtung].MoveX );            { Bewegung in neue Richtung }
      inc( Y, BewDa[Richtung].MoveY );
      TextColor( 7 );                       { Zeichen wieder sichtbar machen }
      draw;                                  { Objekt an neuer Pos. zeichnen }
    end;
end;

{== Methoden des Objekts Rechteck ==========================================}

{****************************************************************************
*  RECHTECK.RINIT : initialisiert ein Objekt vom Typ RECHTECK, indem es die  *
*                   Position des Objekts sowie die Länge der beiden Seiten   *
*                   speichert.                                               *
**__________________________________________________________________________**
*  Eingabe : X, Y = Koordinate des Objekts                                   *
*            dX   = Seitenlänge horizontal                                   *
*            dY   = Seitenlänge vertikal                                     *
*  Ausgabe : keine                                                           *
****************************************************************************}
procedure rechteck.rinit( XPos, YPos, dX, dY : integer);
begin
  self.XLen := dX;                                     { Seitenlängen merken }
  self.YLen := dY;
  self.pinit( XPos, YPos );                         { Objekt initialisieren }
end;

{****************************************************************************
*  RECHTECK.CHECK : stellt fest, ob sich das übergebene Objekt vom Typ RECHT-*
*                   ECK noch innerhalb des Bewegungsbereichs der Objekte auf *
*                   dem Bildschirm befindet.                                 *
**__________________________________________________________________________**
*  Eingabe : NeuX, NeuY = Ausgangsposition für das zu überprüfende Recht-    *
*  Ausgabe : TRUE, wenn das Objekt noch innerhalb des Bewegungsbereichs      *
*            liegt, sonst FALSE.                                             *
*  Info    : Der Test wird durchgeführt, indem die Gültigkeit der Koor-      *
*            dinaten der vier Ecken des Rechtecks mit Hilfe der Methode      *
*            PUNKT.CHECK überprüft wird.                                     *
****************************************************************************}
function rechteck.check( NeuX, NeuY : integer ) : boolean;
begin
  check := inherited self.check( NeuX, NeuY )                            and
           inherited self.check( NeuX+self.XLen-1, NeuY )                and
           inherited self.check( NeuX+self.XLen-1, NeuY+self.YLen-1 )    and
           inherited self.check( NeuX, NeuY+self.YLen-1 );
end;

{****************************************************************************
*  RECHTECK.DRAW : mal ein Objekt vom Typ RECHTECK an seiner augenblick-     *
*                  lichen Position auf dem Bildschirm                        *
**__________________________________________________________________________**
*  Eingabe : keine                                                           *
*  Ausgabe : keine                                                           *
*                                                                            *
*  Info    : Diese Methode wird von der Methode MOVE aufgerufen, um das      *
*            Objekt auf dem Bildschirm zu zeichnen oder vom Bildschirm zu    *
*            entfernen.                                                      *
****************************************************************************}
procedure rechteck.draw;

var i : byte;                                               { Schleifenzähler }

begin
  with self do                          { auf die Felder des Objekts zugreifen }
    begin
      {— obere Linie zeichnen ———————————————————————————————————————————}
      GotoXY( self.GetX, self.GetY ); Write('┌');
      for i := 1 to XLen-2 do write( '-' ); Write('┐');

      {— senkrechte Linien zeichnen —————————————————————————————————————}
      for i := 1 to YLen - 1 do
        begin
          GotoXY( self.GetX, self.GetY + i ); write( '|' );
          GotoXY( self.GetX + XLen - 1, self.GetY + i ); write( '|' );
        end;

      {— untere Linie ziehen ————————————————————————————————————————————}

      GotoXY( self.GetX, self.GetY + YLen - 1 );  Write('└');
      for i := 1 to XLen-2 do write( '-' ); Write('┘');
    end;
end;

{== Methoden des Objekts Dreieck ===========================================}

{****************************************************************************
*  DREIECK.DINIT : initialisiert ein Objekt vom Typ DREIECK, indem es die    *
*                  Position des Objekts sowie die Länge der Grundseite       *
*                  speichert.                                                *
**__________________________________________________________________________**
*  Eingabe : X, Y = Koordinate des Objekts                                   *
*            dS   = Länge der Grundseiten                                    *
*            dY   = Seitenlänge vertikal                                     *
*  Ausgabe : keine                                                           *
*  Info    : die Länge der beiden Schenkel entspricht der halben Länge der   *
*            Grundseite.                                                     *
****************************************************************************}
procedure dreieck.dinit( XPos, YPos, dS : integer);
begin
  self.SLen := dS or 1;                      { Seitenlänge muß ungerade sein }
  self.pinit( XPos, YPos );                          { Objekt initialisieren }
end;

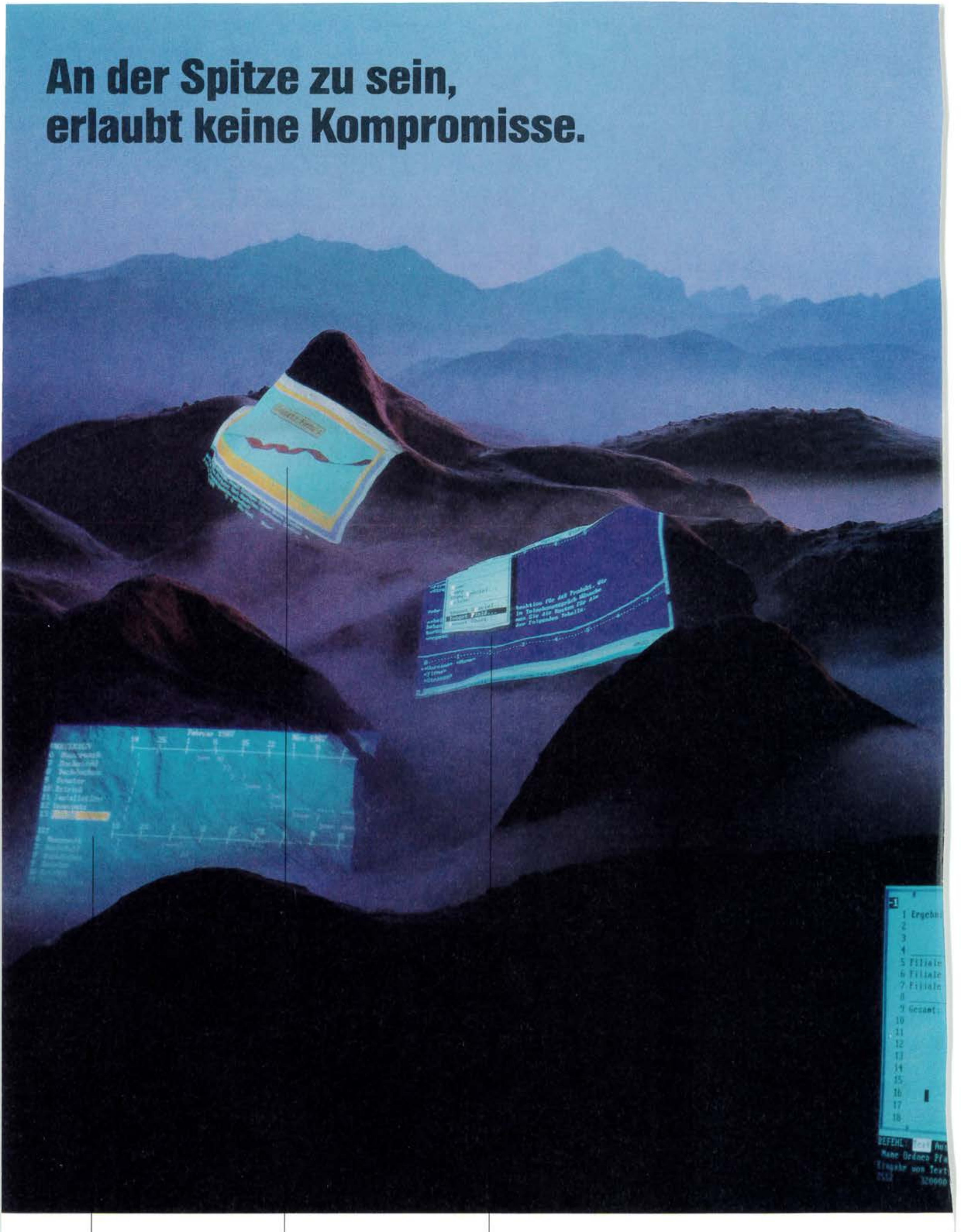
{****************************************************************************
*  DREIECK.CHECK : stellt fest, ob sich das übergebene Objekt vom Typ DREI-  *
*                  ECK noch innerhalb des Bewegungsbereichs der Objekte auf  *
*                  dem Bildschirm befindet.                                  *
**__________________________________________________________________________**
*  Eingabe : NeuX, NeuY = die zu überprüfende Ausgangspositon des Dreiecks   *
*  Ausgabe : TRUE, wenn das Objekt noch innerhalb des Bewegungsbereichs      *
*            liegt, sonst FALSE.                                             *
*  Info    : Der Test wird durchgeführt, indem die Gültigkeit der Koor-      *
*            dinaten der drei Ecken des Dreiecks mit Hilfe der Methode       *
*            PUNKT.CHECK überprüft wird.                                     *
****************************************************************************}
function dreieck.check( NeuX, NeuY : integer ) : boolean;

begin
  check := inherited self.check( NeuX, NeuY )                         and
           inherited self.check( NeuX + self.SLen - 1, NeuY )         and
           inherited self.check( NeuX + ( self.SLen shr 1 ),
                                 NeuY - ( self.SLen shr 1 ) );
end;

{****************************************************************************
*  DREIECK.DRAW : mal ein Objekt vom Typ DREIECK an seiner augenblicklichen  *
*                 Bildschirmosition                                          *
**__________________________________________________________________________**
*  Eingabe : keine                                                           *
*  Ausgabe : keine                                                           *
*  Info    : Diese Methode wird von der Methode MOVE aufgerufen, um das      *
*            Objekt auf dem Bildschirm zu zeichnen oder vom Bildschirm zu    *
*            entfernen.                                                      *
****************************************************************************}
procedure dreieck.draw;

var i : byte;                                               { Schleifenzähler }

begin
  with self do                { erlaubt den Zugriff auf die Felder des Objekts }
    begin
      {— Grundseite malen ———————————————————————————————————————————————}
      GotoXY( self.GetX, self.GetY );
      for i := 1 to SLen do write( '■' );
      {— die beiden Schenkel zeichnen ———————————————————————————————————}
      for i := 1 to ( SLen shr 1 ) do
        begin
          GotoXY( self.GetX + i, self.GetY - i ); write( '*' );
          GotoXY( self.GetX + SLen - i - 1, self.GetY - i ); write( '*' );
        end;
    end;
end;
{===========================================================================}
{==   Hauptprogramm   ======================================================}
{===========================================================================}
var r     : array [1..ANZ_OBJEKTE] of rechteck;          { Arrays mit jeweils }
    d     : array [1..ANZ_OBJEKTE] of dreieck;          { einem Typ von Objekt }
    p     : array [1..ANZ_OBJEKTE] of punkt;
    anzr,                         { Anzahl der Dreiecke, Punkte und Rechtecke }
    anzd,
    anzp ,
    i     : integer;                                       { Schleifenzähler }
    ch    : char;                                   { zum Abfragen der Taste }
begin
  Randomize;                              { Zufallsgenerator initialisieren }
  repeat
    {— Bildschirm aufbauen ——————————————————————————————————————————————}
    TextColor( 7 );  TextBackground( 0 );                   { normale Schrift }
    ClrScr;                                             { Bildschirm löschen }
    TextColor( 0 );  TextBackground( 7 );                   { inverse Schrift }
    ClrEol;                                { erste Bildschirmzeile invertieren }
    write( '     OOPDEMO - demonstriert die objektorientierten Features',
           ' von QuickPascal' );
    GotoXY( 1, 25 );                       { Cursor in letzte Bildschirmzeile }
    ClrEol;                                        { und auch die invertieren }
    write( ' SPACE = neuer Durchlauf | RETURN = Ende             ',
           '(c) 1989 by Michael Tischer' );
    TextBackground( 0 );                   { Hintergrundfarbe auf schwarz setzen }
    anzr := 0;                                { es gibt bisher weder Punkte, }
    anzd := 0;                                { noch Kreise, noch Dreiecke   }
    anzp := 0;
    {— Objekte zufällig erzeugen ————————————————————————————————————————}
    for i := 1 to ANZ_OBJEKTE do                       { ANZ_OBJEKTE erzeugen }
      case Random(3) of                              { Art des Objekts auslosen }
        0 : begin                                           { Punkt erzeugen }
              inc( anzp );                      { Anzahl Punkte inkrementieren }
              new( p[anzp ] );                             { Objekt erzeugen }
              p[ anzp ].pinit( Random(79)+1, Random(24)+2 );
            end;
        1 : begin                                        { Rechteck erzeugen }
              inc( anzr );                   { Anzahl Rechtecke inkrementieren }
              new( r[ anzr ] );                            { Objekt erzeugen }
              r[ anzr ].rinit( Random(60)+1, Random(15)+2,
                               Random(10)+3, Random(5)+2 );
            end;
        2 : begin                                         { Dreieck erzeugen }
              inc( anzd );                    { Anzahl Dreiecke inkrementieren }
              new( d[ anzd ] );                            { Objekt erzeugen }
              d[ anzd ].dinit( Random(60)+1, Random(10)+14, Random(7)+3 );
            end;
      end;
    {— Objekte bewegen, bis eine Taste betätigt wird ————————————————————}
    ch := #0;                                     { noch keine Taste betätigt }
    repeat
      for i := 1 to ANZ_OBJEKTE do                   { die Objekte durchlaufen }
        begin
          if ( i <= anzp ) then                   { gibt es noch einen Punkt? }
            p[ i ].move;                                 { Ja, Punkt bewegen }
          if ( i <= anzr ) then                  { gibt es noch ein Rechteck? }
            r[ i ].move;                              { Ja, Rechteck bewegen }
          if ( i <= anzd ) then                   { gibt es noch ein Dreieck? }
            d[ i ].move;                               { Ja, Dreieck bewegen }
        end;
      if KeyPressed then                        { wurde eine Taste betätigt? }
        begin
          ch := ReadKey;                              { betätigte Taste holen }
          if ( ch = #0 ) then                    { erweiterter Tastaturcode? }
            begin                       { Ja, Taste holen aber nicht verwerten }
              ch := ReadKey;
              ch := #0;                                      { Taste wegwerfen }
            end;
        end;
    until ( ch = ' ' ) or ( ch = #13 );
    {— die erzeugten Objekte wieder löschen —————————————————————————————}
    for i := 1 to ANZ_OBJEKTE do                     { die Objekte durchlaufen }
      begin
        if ( i <= anzp ) then                     { gibt es noch einen Punkt? }
          dispose( p[ i ] );                                   { Ja, löschen }
        if ( i <= anzr ) then                    { gibt es noch ein Rechteck? }
          dispose( r[ i ] );                                   { Ja, löschen }
        if ( i <= anzd ) then                     { gibt es noch ein Dreieck? }
          dispose( d[ i ] );                                   { Ja, löschen }
      end;
  until ( ch = #13 );                     { wiederholen, bis Return betätigt wurde }
  ClrScr;                                               { Bildschirm löschen }
end.
```



MICROSOFT PROJECT
Projektplanung und Projektüberwachung.
MS/DOS

MICROSOFT CHART
Grafik-Programm.
MS/DOS

MICROSOFT WORKS
Die Grundausstattung für Ihren PC.
MS/DOS

Perfekte Software ohne Kompromisse braucht gerade heute »Optionen auf die Zukunft«. An die Stelle von kurzfristig passenden Einzellösungen müssen ausgebaute und ausbaubare Konzepte treten. Neben dem perfekten Einzelprodukt muß die jederzeit realisierbare, datenkompatible Ausweitung in ein System stehen, damit die Software-Lösungen mit den Software-Anforderungen eines Unternehmens wachsen können.

Microsoft hat deshalb von Anfang an in einer kompletten Applikationen-Familie gedacht und diese konsequent weiterentwickelt. Dabei haben Mitglieder wie MICROSOFT WORD oder MICROSOFT MULTIPLAN den Standard ihrer Klasse gesetzt. Dies bestätigt zusätzlich die Marktentsprechung dieses Konzeptes, mit dem die Marke Microsoft sich von anderen differenziert.

MICROSOFT WORD, MICROSOFT MULTIPLAN, MICROSOFT CHART und MICROSOFT PROJECT haben die gleiche Bedieneroberfläche und eine einheitliche Tastatur- oder Mausbedienung. Alle tauschen untereinander perfekt Daten aus und unterstützen auch die technologisch neuesten Peripheriegeräte. Microsoft Applikations-Software ist für alle angebotenen MS-DOS Personal Computer geeignet. Vom 8088-Laptop bis zum 486-Top-Modell. Alle Produkte sind in PC-Netzen einsetzbar. Mit MICROSOFT MULTIPLAN 4.0 sowie MICROSOFT WORD 5.0 stehen darüber hinaus die ersten echten OS/2-Anwendungen von Microsoft zur Verfügung.

Damit bedeutet die Entscheidung für Microsoft Applikationen den Einstieg in ein »offenes System«, dessen Integrationsgrad und jeweiliger Leistungs-Umfang vom Nutzer oder Entscheider bestimmt werden, statt von Software-Engpässen. Und das – so meinen wir – ist eine der Freiheiten, die man sich leisten sollte. Damit Ihre Software jetzt und in Zukunft alles leisten kann.



ZUKUNFT DER SOFTWARE

MICROSOFT MULTIPLAN
Tabellenkalkulationsprogramm.
MS/DOS MS/OS/2

MICROSOFT WORD
Textverarbeitungsprogramm.
MS/DOS MS/OS/2

OOP unter QuickPascal und Turbo Pascal im Vergleich

Es sind in der letzten Zeit mehrere Artikel über die objektorientierten Möglichkeiten von QuickPascal und Turbo Pascal erschienen. Viele dieser Artikel enthielten technische und sachliche Fehler. So wurde beispielsweise in einigen Artikeln behauptet, daß QuickPascal keine virtuellen Methoden enthält, während es tatsächlich so ist, daß in QuickPascal alle Methoden virtuell sind. In einigen Artikeln wurde das frühe und das späte Binden (»early binding«/ »late binding«) besprochen und es wurde dort behauptet, daß das späte Binden zur Laufzeit irgendeine zeitaufwendige Suche notwendig macht; dies ist nicht so.

In diesem Text werden mehrere technische Themen besprochen, die beim Vergleich von Turbo Pascal und QuickPascal wichtig sind. Die Absicht dieses Artikels ist es, Fehler zu korrigieren und klarzustellen, worin die Unterschiede zwischen den objektorientierten Erweiterungen dieser beiden Produkte bestehen.

Dynamische oder statische Objekt-Allokierung

Dies ist sehr einfach; es entspricht völlig der Deklaration von Variablen oder Zeigern auf Variablen. Bei der traditionellen Programmierung ist es besser, ein Datum mit variabler Größe, ein sehr großes Datum oder eines mit unbestimmter Größe dynamisch zu allokieren. Anstatt eine Variable zu deklarieren, deklariert man einen Zeiger auf eine Variable und »dereferenziert« ihn, wenn erforderlich.

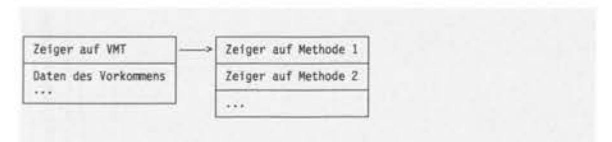
Bei der Objektoriertierten Programmierung (OOP) kann der Platz für die Daten eines Objekts statisch im Daten- oder Stacksegment allokiert werden (statische Allokierung) oder man kann das Objekt dynamisch auf dem Heap allokieren (dynamische Allokierung). Es ist zu beachten, daß hier die Terminologie etwas irreführend ist, da »statische« Objekte, die lokal in einer Prozedur deklariert werden, nicht in demselben Sinn »statisch« sind, wie es globale Variablen sind. Wie lokale Variablen, werden lokale »statische« Objekte beim Eintritt in eine Prozedur auf dem Stack allokiert und beim Verlassen der Prozedur wieder deallokiert.

Der Vorteil dynamischer Objekte besteht darin, daß sie keinen Platz im Daten- oder Stacksegment benötigen (dort ist sowieso nur begrenzt Platz) und sie können nach Bedarf allokiert und deallokiert werden. Der Nachteil dynamischer Objekte besteht darin, daß sie beim Zugriff immer dereferenziert werden müssen, sie sind also langsamer.

Auf statische Objekte kann man schneller zu greifen, sie belegen jedoch Speicher im Stack- und Datensegment. Es ist darüber hinaus sehr wahrscheinlich, daß gut strukturierte Programme, die ausgiebig Prozeduren und Funktionen verwenden, die Vorteile statischer Objekte verringern werden. (Sobald eine statisches Objekt als ein Argument einer Prozedur übergeben wird, wird es im Prinzip, besonders aus Performance-Sicht, ein dynamisches Objekt.)

Virtuelle Methoden

C++ und Turbo Pascal 5.5 erlauben sowohl virtuelle als auch nicht virtuelle Methoden; in QuickPascal sind alle Methoden virtuell. Virtuelle Methoden sind Methoden, auf die indirekt über eine »virtuelle Methoden-Tabelle« (eine Sprungtabelle) zugegriffen wird. Ein Objekt, daß virtuelle Methoden verwendet, sieht im Speicher so aus:



Die Struktur der VMT wird von der verwendeten Klassenhierarchie bestimmt. Ein Beispiel in QuickPascal:

```
{ QP Code }
type
        shape = object
                        procedure draw;
                        procedure erase;
                end;

        cube = object(shape)
                        procedure draw; override;
                        procedure rotate;
                end;
```

Die virtuelle Methoden-Tabelle für »shape« hat zwei Einträge: einen Zeiger auf den Code der Prozedur »draw« und einen Zeiger auf den Code der Prozedur »erase«. Die virtuelle Methoden-Tabelle für »cube« hat drei Einträge: einen für »draw«, einen für »erase« und einen für »rotate« (in dieser Reihenfolge). In »cube« zeigt der Eintrag für »erase« auf genau den gleichen Programmcode, wie in der Tabelle für »shape«, da dieser Code übernommen wurde (inherited). Der Eintrag für »draw« zeigt auf neuen Programmcode, da dieser übergangen wurde (override). Der Eintrag für »rotate« besteht ebenfalls aus neuem Code, da es sich hier um eine hinzugefügte Methode handelt. Die Struktur der VMT wurde übernommen (inherited). Die beiden ersten Einträge von »cube« (und jeder Unterklasse von »shape« oder »cube«) sind immer Zeiger auf die Prozeduren »draw« und »erase«. Der Vorteil davon ist, daß man dies schreiben kann:

```
var
        s : shape;
        c : cube;
```

```
        procedure ReDrawAThing( o : shape );
        begin
                o.Erase;
                o.Draw;
        end;
begin
        ...
        ReDrawAThing( s );
        ReDrawAThing( c );
        ...
end;
```

Da auf virtuelle Methoden immer indirekt über die VMT zugegriffen wird, ruft die Prozedur ReDrawAThing immer den »richtigen« »draw«- und »erase«-Code auf. Wenn im obigen Beispiel »draw« und »erase« keine virtuellen Methoden wären, würde der Code von ReDrawAThing die Tatsache ignorieren, daß »cube« die »draw«-Methode übergangen hat und immer den »draw«-Code von »shape« aufrufen, da der Parameter »o« als Objekt vom Typ »shape« deklariert wurde.

Die Vorteile virtueller Methoden sind die wesentlich bessere Wiederverwendbarkeit von Code und wesentlich logischeres Verhalten. Im obigen Beispiel macht es Sinn, daß der neue Methodencode aufgerufen wird, da der Code für »draw« in der »cube«-Klasse übergangen wurde. Die ausgiebige Verwendung virtueller Methoden ist in gut entworfenem, objektorientiertem Code häufig.

Der Nachteil virtueller Methoden ist der langsamere Zugriff. Nicht virtuelle Methoden werden in einfache Aufrufe ohne Indirektion aufgelöst.

Konstruktoren und Destruktoren

Konstruktoren und Destruktoren sind besondere Methoden, die bei der Initialisierung und Beendigung eines Objekts behilflich sind. In C++ sind die Konstruktoren und Destruktoren sehr leistungsfähige Bestandteile. Der Compiler ruft die Konstruktoren und Destruktoren automatisch auf, wenn Objekte deklariert und wenn sie beendet werden. Bei statischen Objekten in einer Prozedur (und als solche werden sie auf dem Stack allokiert) kann dies sehr angenehm sein, da C++ die Destruktoren von Objekten aufruft, sobald die Objekte ihren Gültigkeitsbereich verlassen (z.B. wenn die Prozedur verlassen wird). Dieser automatische Aufruf ist ein wichtiger Aspekt der Konstruktoren und Destruktoren in C++.

Turbo Pascal 5.5 verwendet die Konstruktoren/Destruktoren-Terminologie, implementiert in Wirklichkeit jedoch die Funktionalität von C++ nicht. In Turbo Pascal müssen Konstruktoren explizit aufgerufen werden und unterscheiden sich nur wenig von normalen Methoden. Die beiden einzigen Eigenheiten der Konstruktoren in Turbo Pascal 5.5 sind:

1. Man muß einen Konstruktor aufrufen, um die virtuelle Methoden-Tabelle für ein Objekt einzurichten, das virtuelle Methoden verwendet.
2. Konstruktoren können als Teil von »new« aufgerufen werden:

```
{ TP 5.5 code }
type
        point = object
                x, y : integer;
                constructor init;
                procedure draw; virtual;
        end;
var
        pp : ^point;
begin
        new( pp, init);
end;
```

Die Destruktoren von Turbo Pascal 5.5 sind im Vergleich zu C++ ähnlich eingeschränkt. In Turbo Pascal 5.5 besteht der einzige Unterschied zwischen einer Destruktor-Methode und einer normalen Methode darin, daß ein Destruktor bei »dispose« aufgerufen werden kann:

```
{ TP 5.5 code }
type
        point = object
                x, y : integer;
                constructor init;
                destructor done;
                procedure draw; virtual;
        end;
        point3d = object(point)
                z : integer;
                destructor done;
        end;
var
        pp : ^point;
        p3p : ^point3d;

procedure goaway( var p : point);
begin
        dispose( p, done);
end;

begin
        new( pp, init);
        new( p3p, init);
        goaway( pp );
        goaway( p3p );
end;
```

Das Besondere an den Destruktoren von Turbo Pascal 5.5 ist, daß sie, wenn sie als Teil von »dispose« aufgerufen werden, die Anzahl der zu deallokierenden Bytes übergeben. Im obigen Beispiel deallokiert die Prozedur »goaway« die richtige Anzahl Bytes, da die Destruktoren dem »dispose«-Aufruf diese Information übergeben.

QuickPascal verfügt nicht über Konstruktoren und Destruktoren, da es sie nicht braucht. QuickPascal richtet die virtuelle Methoden-Tabelle automatisch ein, wenn ein Objekt allokiert wird. Der »dispose«-Aufruf in QuickPascal deallokiert automatisch die richtige Anzahl Bytes wenn ein Objekt freigegeben wird (sogar in einer Prozedur wie »goaway«, in der das Objekt als Parent deklariert wurde). Der »dispose«-Aufruf weiß die richtige Anzahl Bytes, die deallokiert werden müssen, da diese Information als Teil der virtuellen Methoden-Tabelle gespeichert wird. In QuickPascal könnte das obige Beispiel so codiert werden:

```
{ QP code }
type
        point = object
                x, y : integer;
                procedure init;
                procedure done;
                procedure draw;
            end;
        point3d = object(point)
                z : integer;
                procedure done; override;
            end;
var
        pp : point;
        p3p : point3d;

procedure goaway( var p : point);
begin
        dispose( p );
end;

begin
        new( pp );
        pp.init;
        new( p3p );
        p3p.init
        goaway( pp );
        goaway( p3p );
end;
```

Binde-Strategie und Performance

Als »Binden« wird bezeichnet, wie ein Compiler einen Methoden-Aufruf in wirkliche Maschinenbefehle umsetzt, die den entsprechenden Code aufrufen. Das Binden wird von zwei Dingen beeinflußt: statische oder dynamische Objekt-Allokierung und virtuelle oder nicht virtuelle Methoden. Wir wollen zunächst untersuchen, wie auf die Daten eines Objekts zugegriffen wird.

Bei der statischen Objekt-Allokierung werden Objekte im Daten- oder Stacksegment allokiert, abhängig davon, ob es sich um globale oder lokale Objekte handelt:

```
aCube   | Speicher für Objekt |
        | ...                 |
```

Im Fall eines globalen statischen Objekts ist der Name des Objekts eine direkte Speicherreferenz. Der Zugriff auf die Daten des Objekts ist ein direkter Speicherzugriff, der keine Indirektion erfordert. In Assembler würde das etwa so aussehen:

```
MOV     AX, Datenadresse
```

Dies ist identisch mit dem Zugriff auf globale Variablen.

Der Zugriff auf Daten bei einem dynamischen Objekt erfordert eine Ebene der Indirektion. Im Speicher sieht ein dynamisches Objekt etwa so aus:

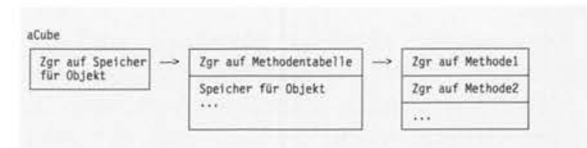
```
aCube  | Zeiger auf Speicher |        | Speicher für Objekt |
       | für das Objekt      |  --->  | ...                 |
```

Um also auf eine Vorkommen-Variable eines dynamischen Objekts zuzugreifen, muß in Assembler etwa folgendes gemacht werden:

```
LES   DI, aCube
MOV   AX, ES:[DI+Datenoffset]
```

Es ist zu beachten, daß in beiden Fällen, sowohl bei statischen als auch bei dynamischen Objekten, zur Laufzeit keine Suche erforderlich ist. In beiden Fällen kann der Compiler beim Kompilieren alle Informationen feststellen, die er wissen muß: bei statischen Objekten die gewünschte Datenadresse, bei dynamischen Objekten den Offset der Daten im Speicher des Objekts.

Der Aufruf einer Methode eines Objekts wird davon beeinflußt, ob die Methode virtuell ist oder nicht. Wenn die Methode virtuell ist, muß die Adresse der Methode aus der Methoden-Sprungtabelle herausgesucht werden. (Im folgenden Beispiel wird angenommen, daß das Objekt dynamisch ist.)



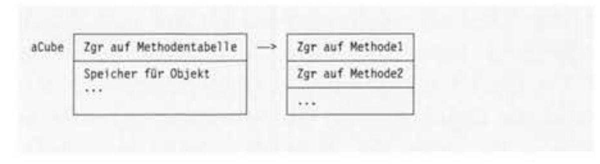
(Anmerkung: Alle Objekte einer Klasse zeigen genau auf die gleiche Methoden-Sprungtabelle.)

Für den Aufruf einer virtuellen Methode bei einem dynamischen Objekt ist folgendes notwendig:

```
LES   DI, aCube            ;Adresse des Objektspeichers
PUSH  ES                   ;Zeiger auf sich selbst pushen
PUSH  DI
LES   DI, ES:[DI]          ;Adresse
CALL  FAR ES:[DI+Offset_des_Methodenvektors]
```

Es ist wieder zu beachten, daß selbst bei virtuellen Methoden und dynamischen Objekten zur Laufzeit keine »Suche« notwendig ist. Bereits bei der Kompilierung kann der Compiler den Offset der gewünschten Methode feststellen und den entsprechenden Code erzeugen.

Beim Aufruf einer virtuellen Methode bei einem statischen Objekt gibt es einfach nur eine Ebene der Indirektion weniger:



Unter der Voraussetztung, daß sich das statische Objekt im Datensegment befindet:

```
MOV   DI, [aCube+0]    ; Adresse der Sprungtabelle
PUSH  DS               ; Zeiger auf sich selbst pushen
LEA   AX, aCube
PUSH  AX
CALL  FAR [DI+Offset_des_Methodenvektors]
```

Der Aufruf einer nicht virtuellen Methode bei einem statischen Objekt ist einfach. Der Aufruf wird zu einem einfachen Prozeduraufruf:

```
PUSH    DS              ; Zeiger auf sich selbst pushen
LEA     AX, aCube
PUSH    AX
CALL    Adresse_der_Methode
```

Ganz ohne Frage beeinflussen die Objekt-Allokierungs-Strategie (statisch oder dynamisch) und die Methoden-Zugriffs-Strategie (virtuell oder nicht) die Performance des kompilierten Codes. Da leistungsfähige Anwendungen dazu tendieren, dynamische Allokierung und virtuelle Methoden zu verwenden, müßte die Performance objektorientierter Anwendungen theoretisch schlechter sein, als die traditioneller Anwendungen. In der Praxis ist die Performance jedoch nur selten ein Thema. C++ und objektorientierte Erweiterungen von Pascal bieten eine gute Mischung traditioneller und objektorientierter Techniken. Dadurch kann der Benutzer eine klare Trennlinie ziehen, wo objektorientierte und wo traditionelle Techniken eingesetzt werden. Typischerweise werden objektorientierte Techniken für die Erledigung von Dingen auf übergeordneter Ebene eingesetzt, wie:

- Benutzerschnittstellen,
- Gerätesimulation,
- komplexe Datenstrukturen,
- grafische Objekte.

Diese Dinge arbeiten zumeist nur mit »Benutzergeschwindigkeit«. (Die Anlage eines Fensters braucht nur so schnell zu arbeiten, wie der Benutzer Fenster anlegen kann, das heißt für einen Computer nicht besonders schnell.)

Die traditionellen Techniken werden für systemnähere Operationen verwendet, die einen so geringen Overhead wie möglich erfordern.

Die Performance von C++ und objektorientierten Pascal-Versionen sollte nur mit »reinen« objektorientierten Systemen wie SmallTalk und Actor verglichen werden, in denen selbst Daten auf niedriger Ebene (Konstanten, Integer, Strings, Arrays usw.) als Objekte implementiert sind.

Zusammenfassung

Turbo Pascal 5.5 bietet einige Dinge, die QuickPascal 1.0 nicht hat. Die wichtigsten sind:
1. statisch allokierte Objekte und
2. nicht virtuelle Methoden.

Dadurch werden Perfomance-Vorteile auf Kosten anderer Dinge erzielt: Der Verlust von Daten- oder Stackspeicher im Fall statisch allokierter Objekte oder der Verlust der Wiederverwendbarkeit von Code im Fall der nicht virtuellen Methoden. In der Praxis sieht es so aus, daß nicht triviale, objektorientierte Programme zumeist virtuelle Methoden deklarieren und viele dynamisch allokierte Objekte verwenden. Die QuickPascal-Programmierumgebung wurde in QuickPascal geschrieben und macht ausgiebigen Gebrauch von seinen objektorientierten Möglichkeiten. Die Performance dynamischer Objekte und virtueller Methoden war also ein wichtiges Thema bei der Entwicklung.

Der Einsatz von Konstruktoren und Destruktoren in Turbo Pascal 5.5 ist schlecht gelöst. Erstens wird zwar die Terminologie von C++ verwendet, die Implementierung ist jedoch wesentlich weniger leistungsfähig. Insbesondere das Fehlen des automatischen Aufrufs bei der Anlage eines Objekts und das Löschen (besonders bei statischen Objekten) ist enttäuschend. Zum zweiten scheint die Notwendigkeit von Konstruktoren und Destruktoren in Turbo Pascal ein Ergebnis einer schwachen Implementierung zu sein, weniger der Wunsch, neue Funktionalität zu bieten. Turbo Pascal 5.5 benötigt Konstruktoren und Destruktoren um:
1. die virtuelle Methodentabelle einzurichten und
2. den »dispose«-Aufruf über die Objektgröße zu informieren.

QuickPascal erzielt diese beiden Punkte ohne zusätzliche konzeptionelle Belastungen. Virtuelle Methoden-Tabellen werden automatisch bei der Allokierung eines Objekts eingerichtet. Die »dispose«-Funktion weiß automatisch die richtige Größe eines Objekts.

Lieferengpässe beim Ashton Tate/Microsoft SQL-Server beseitigt

Nachdem die erste Auflage der seit Mai von Ashton Tate ausgelieferten Endanwender-Version des Ashton Tate/Microsoft SQL-Servers überraschend schnell vergriffen war, konnten die entstandenen Lieferengpässe nach Mitteilung von Ashton Tate jetzt behoben werden. Dieses Produkt, das Ergebnis einer gemeinsamen Entwicklung von Microsoft, Sybase und Ashton Tate, ist ab sofort zum Preis von DM 5.950,– zuzüglich Mehrwertsteuer wieder lieferbar.

Mit dem SQL-Server bieten Microsoft und Ashton Tate ein relationales Datenbank-Server-Programm für lokale Netzwerke. Es arbeitet auf der Basis der Client-Server-Architektur, die MS-DOS, MS OS/2 und Microsoft Windows Applikationen einen Zugriff auf Multiuser-Datenbank-Dienstleistungen im gesamten Netzwerk ermöglicht. Der SQL-Server stellt einen erheblichen Fortschritt für die Workstation-orientierte Datenverarbeitung dar. Mit einer breiten Palette von kommerziellen Front-end-Programmen können komplexe transaktionsorientierte Datenverarbeitungs-Systeme realisiert werden. Der SQL-Server hat eine offene Software-Plattform für eine breite Palette von PC-gestützten Programmen (z.B. dBase IV, Tabellenkalkulationen u.a.) geschaffen. Auf PC-LAN-Basis wird damit eine ganz neue Klasse von Datenbanklösungen möglich, die es bisher nur auf Mini- oder Großcomputern gab.

Der Auslieferung des Endanwender-Produktes ging ein sehr erfolgreiches Aktions-Programm voran, in dessen Rahmen Microsoft und Ashton Tate mehr als 1.000 SQL-Server Network Developer's Kits (NDK) an Software-Entwickler lieferten.

Weit verbreitete Programme, die derzeit an den SQL-Server angepaßt werden, sind zum Beispiel DataEase, Advanced Revelation, PC Focus, Paradox, Lotus 1-2-3, dBase IV, Microsoft Excel und viele weitere Produkte. Applikations-Entwickler können darüber hinaus Standard-Programmiersprachen wie C und Cobol nutzen, um Applikationen für den SQL-Server zu entwickeln.

Voraussetzung für den Betrieb des SQL-Servers ist die Installation von MS OS/2. Die Netzwerk-Unterstützung umfaßt Netzwerk-Software wie 3Com 3+Open, IBM LAN-Server, Ungermann-Bass Net/One sowie MS OS/2 LAN-Manager. Zukünftig wird der SQL-Server auch mit anderen verbreiteten PC-Netzwerken, einschließlich solchen auf Basis des Microsoft LAN-Manager und auf Novell NetWare arbeiten.

Lizenzinhaber des SQL-Server Network Developer's Kits haben die Möglichkeit, die SQL Endanwender-Version zu einem günstigen Upgrade-Preis von DM 990,– zzgl. MwSt. bei Ashton Tate zu beziehen.

Neue Version 2.1 der MS-DOS CD-ROM-Extension unterstützt Interleaved-Audio

Auf der vierten CD-ROM-Konferenz in Anaheim, Kalifornien, hat Microsoft kürzlich die Version 2.1 der MS-DOS CD-ROM-Extension vorgestellt. Die neue Version unterstützt die Möglichkeit, neben optischen auch akustische Informationen (Interleaved-Audio) in CD-ROM Extended Architecture (XA) Applikationen zu verarbeiten. Weitere Merkmale sind die Unterstützung der Betriebssystem-Version MS-DOS 4.0 sowie die Möglichkeit, CD-ROM-Laufwerke in einem lokalen Netzwerk einzusetzen.

Vor dem Hintergrund, daß MS-DOS 4.0 in immer stärkerem Maße durch PC-Hersteller ausgeliefert wird, bedeutet die Unterstützung für diese Betriebssystem-Version, daß Software-Entwickler nun damit beginnen können, MS-DOS- und Microsoft Windows-Applikationen zu erstellen, die sich durch eine verbesserte Audiotechnik auszeichnen. Die Verfügbarkeit einer Standard-Schnittstelle zu CD-ROM-XA-Interleaved-Files und Compressed Audio ist ein großer Schritt nach vorne. Es sind nun komplette Multimedia-Software-Applikationen möglich, die eine wesentliche Rolle bei der zukünftigen PC-Datenverarbeitung spielen werden.

Für den Einsatz der MS-DOS CD-ROM-Extension ist ein PC mit MS-DOS Betriebssystem, Version 3.1 oder höher, erforderlich. Die MS-DOS CD-ROM Extension, Version 2.1, wird CD-ROM-Laufwerk-Herstellern im Rahmen von Lizenzverträgen durch Microsoft ab sofort zur Verfügung gestellt.

Microsoft Excel unterstützt IBM OfficeVision/2

Microsoft kündigte die Entwicklung von Applikationssoftware zur Integration in das neue Paket IBM OfficeVision/2 an. Anläßlich der IBM-Vorstellung von OfficeVision/2 gab Microsoft einen Ausblick auf die zukünftige Presentation Manager Version des weitverbreiteten Tabellenkalkulations-Programms Microsoft Excel. Diese Version wird, so das Unternehmen, die erste Applikationssoftware von Microsoft sein, die die Vorteile von MS OS/2 1.1 und 1.2 nutzt. Microsoft plant, eine komplette Reihe von Applikationen mit der grafischen Bedieneroberfläche des Presentation Managers herauszubringen.

Nach den Worten von Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, entspricht Microsoft damit einer Forderung der Anwender, »die bei allen eingesetzten Applikationen eine einheitliche

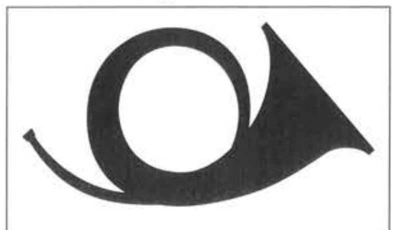
Bedieneroberfläche und die Übereinstimmung zum SAA-Standard der IBM wünschen«.

Excel ist das führende Tabellenkalkulations-Programm für grafikorientierte Betriebssystemumgebungen. Microsoft Excel unter Windows und für den Apple Macintosh sind bereits verfügbar. Die Version für MS OS/2 1.1 und MS OS/2 1.2 soll noch im Laufe dieses Jahres ausgeliefert werden. Das Programm umfaßt leistungsfähige Analyse-Werkzeuge sowie Funktionen zur Organisation und Präsentation von Daten.

Microsoft Excel kann sowohl Informationen von IBM OfficeVision/2 empfangen als auch dorthin senden. Mit Hilfe einer Dialog-Box, die in der leistungsfähigen Makro-Programmiersprache von Microsoft Excel erstellt wurde, hat der Anwender die Möglichkeit, einen Befehl über die Mail-Funktion von OfficeVision/2 an eine andere Workstation zu schicken. Ist dort ebenfalls Microsoft Excel für den Presentation Manager installiert, nimmt dieses Programm den Befehl entgegen, führt ihn aus und schickt mittels OfficeVision/2 eine Bestätigung an den Absender. Diese Art der Integration wird möglich, da Microsoft Excel die Service-Funktionen von OfficeVision/2 und die Vorteile von IBM Personal System/2 unter der erweiterten Version des Betriebssystems MS OS/2 nutzt.

Deutsche Bundespost entscheidet sich für MS OS/2

Einen entscheidenden Durchbruch konnte der neue Betriebssystemstandard MS OS/2 von Microsoft und IBM nun auch im Bereich der öffentlichen Verwaltung erzielen. Nach ersten Erfolgen dieses MS-DOS Nachfolgesystems in der Privatindustrie, hat sich nun der größte deutsche Arbeitgeber, die Deutsche Bundespost, für MS OS/2 entschieden. Für die umfassende Modernisierung und Elektronisierung ihrer Dienstleistungen sollen sämtliche Schalter in allen bundesdeutschen Postämtern mit MS OS/2-Personalcomputern ausgerüstet werden. Alle schalterüblichen Dienstleistungen werden dort in EPOS (Electronic Post Office System) zusammengefaßt. Der Lieferant von Hard- und Software wird zur Zeit im Rahmen einer öffentlichen Ausschreibung des Posttechnischen Zentralamtes (PTZ) gesucht.



Derzeit besteht EPOS noch aus 4.000 Nixdorf PCs mit 80186er Intel-Prozessor und einem auf MS-DOS basierenden Betriebssystem. Eine einheitliche Bedieneroberfläche wurde vom PTZ vorgegeben und wird auch unter MS OS/2 beibehalten. Es handelt sich um eine geschlossene Anwendung, bei der der Schalterbeamte keinen Zugriff auf das Betriebssystem hat. Insgesamt werden 280 unterschiedliche Vorgänge – von der Postsparkassenbuch-Verwaltung über Ein- und Auszahlungen bis hin zum Verkauf von Briefmarken und Postkarten – über die EPOS-Computer abgewickelt. Das Kassenbuch wurde durch persönliche »Kassen-Disketten« der einzelnen Schalterbeamten ersetzt.

Das unter Microfocus Cobol entwickelte EPOS-Programm ist rund 2 Mbyte groß, beansprucht jedoch derzeit durch Datenswitching nur 912 Kbyte Hauptspeicher. Die Speicherbegrenzung von MS-DOS war daher auch das Hauptargument für das PTZ, auf MS OS/2 umzusteigen. Anette Klüber-Meyer, Referatsleiterin für Hard- und Grundsoftware für Endgeräte beim PTZ: »EPOS ist durch die hohen Anforderungen an Anwenderfreundlichkeit und Datensicherheit und durch die hohe Komplexität heute an die Grenzen des Betriebssystems MS-DOS gestoßen. Wir haben uns daher nach einem neuen, allgemein anerkannten und leistungsfähigen Standard umgesehen. MS OS/2 bzw. IBM BS/2 entspricht hier voll unseren heutigen und zukünftigen Anforderungen. Wir werden die verbesserte Speicherverwaltung und die Multitaskingfähigkeiten nutzen.«

In der ersten Stufe der Implementierung wird das EPOS-Programm strukturell unverändert auf MS OS/2 portiert. Erste Portierungsversuche mit dem Cobol OS/2-Compiler von Microfocus und Microsoft waren erfolgreich. In dieser Projektphase wird lediglich die gegenüber MS-DOS verbesserte Speicherverwaltung benutzt. Standardrechner im EPOS sollen Industriestandard-PCs mit dem 80386-Prozessor von Intel und 4 Mbyte Hauptspeicher werden. Der vergrößerte Arbeitsspeicher wird die Performance der EPOS-Software wesentlich erhöhen.

In der zweiten Projektphase findet die Integration von Datenfernübertragungsmöglichkeiten in EPOS statt. Unter MS OS/2 werden über Bildschirmtext im Hintergrund z.B. Deckungsabfragen im Postgirodienst abgewickelt.

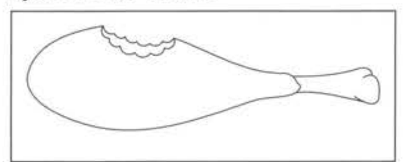
In der dritten Ausbauphase schließlich sollen weitere Multitaskingmöglichkeiten von MS OS/2 für EPOS nutzbar gemacht werden. MS OS/2 ermöglicht die Effektivierung der EPOS-Software durch seine vielfachen Multitasking-Möglichkeiten. Weitere Ausbaumöglichkeiten – etwa die Integration in ISDN – werden als Optionen für die Zukunft beim PTZ derzeit diskutiert.

Microsoft verspricht sich von der Entscheidung der Deutschen Bundespost für MS OS/2 eine deutliche Stärkung dieses neuen Betriebssystemstandards. Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH: »Dies ist weltweit die bislang stückzahlmäßig größte Ausschreibung für ein MS OS/2-

Projekt. Wir betrachten die Entscheidung des größten bundesdeutschen Arbeitgebers und des größten öffentlichen Dienstleisters für MS OS/2 als wegweisend. Neben der Industrie entscheiden sich nun auch immer mehr öffentliche Auftraggeber für MS OS/2 als den derzeit modernsten PC-Betriebssystemstandard.«

Kentucky Fried Chicken »kocht« mit MS OS/2

Die amerikanische Fast-Food-Kette Kentucky Fried Chicken, ein Unternehmen der weltweiten Pepsico Inc., sieht in der »computerunterstützten intelligenten Küche« das richtige Rezept für einen schnelleren Service und bessere Profitabilität. Als »Zutat« setzt das Unternemen jetzt auf das Betriebssystem MS OS/2.



Um einen optimalen Betriebsablauf zu ermöglichen und die Restaurant-Manager bei ihrer Disposition zu unterstützen, installiert Kentucky Fried Chicken ein neuartiges Netzwerk von Personalcomputern und Kassenterminals, das außerdem den Autoschalter-Service um 30 Prozent beschleunigt. Der Start des Projekts erfolgt demnächst in drei Standorten in Louisville.

Neben der Material- und Zeit-Disposition liefern die PCs auch Informationen zu anderen Bereichen der Betriebsführung, wie Arbeitsplanung und Personal-Koordination, Absatzprognosen und aktuelle Preisdaten. Die Anwendung wird auf dem NCR 2760 Food Service System realisiert, das PCs umfaßt, die unter dem Multitasking-Betriebssystemstandard MS OS/2 von Microsoft laufen. In den USA werden mehr als 1.200 Restaurants mit dem neuen System ausgestattet.

Microsoft stellt Version 1.2 des Betriebssystems MS OS/2 mit Presentation Manager vor

Mit der neuen Version 1.2 des Betriebssystems MS OS/2 bietet Microsoft jetzt ein Dateisystem mit noch besseren Leistungsmerkmalen, einschließlich einer verbesserten Presentation Manager-Oberfläche. Die zusätzlichen Funktionen und Neuerungen in MS OS/2 Version 1.2, die ab September 1989 verfügbar sein wird, reflektieren die von Anwendern und Entwicklern geäußerten Forderungen an den neuen Betriebssystemstandard MS OS/2. Mit der Auslieferung von bedeutenden MS OS/2 Applikationen und der verbesserten Version 1.2 schafft Microsoft die Grundlage für eine erfolgreiche Weiterentwicklung im Jahre 1990.

MS OS/2 Version 1.2 beinhaltet ein neues Dateisystem, das die heute vorhandenen technischen Möglichkeiten zur Verbesserung der System-Antwortzeit und der Gesamt-Systemleistung nutzt. Diese neue Technik ist zudem die Basis für weitere Dateisystem-Funktionen, die in späteren Versionen von MS OS/2 verfügbar sein werden. Der größte Nutzen aus dem neuen Dateisystem ergibt sich bei Datenbank- und Server-Applikationen, wo sehr viele Plattenzugriffe stattfinden und große Mengen von Daten geschrieben und gelesen werden. Das neue Dateisystem ist deshalb in erster Linie für den Einsatz auf Festplatten ausgelegt. MS OS/2, Version 1.2, kann bisherige MS-DOS Dateisysteme ebenso unterstützen wie das neue Dateisystem, so daß die Kompatibilität zwischen MS OS/2 und MS-DOS hinsichtlich der auf den Systemen laufenden Applikationen gewahrt bleibt. MS OS/2 1.2 ist aufwärtskompatibel zur Version 1.1.



Software-Entwickler haben daher die Möglichkeit, eine einzige Programmversion zu erstellen, die auf beiden Betriebssystemen läuft und trotzdem alle Vorteile der in MS OS/2 1.2 verfügbaren Funktionen ausnutzt.

Für Borland International, Entwickler des Programms Side-Kick, der ersten Applikation für MS OS/2 mit Presentation Manager, betont Brad Silverberg, Vice President für Forschung und Entwicklung, »daß zukünftige Versionen der Borland-Produkte für MS OS/2 die speziellen Funktionen in MS OS/2 1.2 unterstützen werden«.

Die Verbesserungen der Presentation Manager-Oberfläche in MS OS/2 1.2 beinhalten nun direkte Manipulationsmöglichkeiten von Systemelementen, wie das Kopieren von Dateien durch entsprechendes Verschieben der Symbole mit der Maus und den zusätzlichen Einsatz von Symbolen zur Darstellung von Informationen. Neu ist auch ein Systemeditor, der die Presentation Manager-Schnittstelle in ähnlicher Weise nutzt wie der Notepad-Texteditor in Microsoft Windows. Hinzugekommen sind ferner eine Dual-Boot-Funktion für MS-DOS und MS OS/2 sowie ein PostScript-Gerätetreiber. Die MS-DOS Kompatibilitätsbox, die den Ablauf von MS-DOS Applikationen auf einem MS OS/2-PC ermöglicht, hat in der Version 1.2 einen größeren Adreßraum. Außerdem können MS-DOS Programme auch direkt aus der Presentation Manager-Oberfläche gestartet werden. Die Handhabung wird dadurch wesentlich vereinfacht.

IBM OfficeVision/2 nutzt Microsoft LAN-Manager Technik

Anläßlich der SAA OfficeVision-Ankündigung gab IBM bekannt, daß der Einsatz von Microsoft Netzwerktechnik in drei Bereichen geplant ist: Die erweiterte Version des Betriebssystems IBM OS/2 1.2 (in der BRD BS/2 1.2) wird 13 Kategorien der LAN-APIs (Application Programming Interfaces) unterstützen, die kompatibel sind mit den entsprechenden Microsoft LAN-Manager API-Kategorien. Außerdem nutzt IBM in dieser Produktlinie den IBM LAN Requester, Version 1.2, der auf der LAN-Manager-Technik basiert. Darüber hinaus kündigte IBM die Unterstützung der Version 1.2 für ausgewählte Ethernet-Adapter an, die entsprechend der Microsoft/3Com NDIS (Network Driver Interface Specifications) geschrieben sind.

Die IBM-Ankündigungen stärken die Position des Microsoft LAN-Managers in zwei Schlüsselbereichen: der Entwicklung von Netzwerk-Applikationen und der Verfügbarkeit eines standardisierten LAN-Adapterkarten-Interfaces für Netzwerkprotokoll-Software. Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, betonte, daß für unabhängige Software-Entwickler, Endanwender und auch für Hardware-Hersteller »das Leben nun ein wenig leichter gemacht wird«. In derselben Weise, wie NetBIOS die Entwicklung der ersten Netzwerk-Applikationen gefördert hat, werden die LAN-Manager APIs die Erstellung der nächsten Generation von echten verteilten Applikationen unterstützen.

Die Verwendung der Microsoft LAN-Manager-Technik in SAA OfficeVision bietet dem Anwender eine freie Wahl bei der Netzwerk-Umgebung. Arbeitsplatz-Computer, auf denen SAA OfficeVision-Software unter IBM OS/2 1.2 läuft, können Server-Ressourcen nutzen, die unter dem Microsoft LAN-Manager laufen. Außerdem wird Arbeitsplatz-Computern unter dem LAN-Manager der Zugriff auf den IBM LAN Server ermöglicht.

Die von IBM angekündigte LAN API-Unterstützung für OS/2 1.2, in Verbindung mit der bereits vorhandenen Unterstützung von Named Pipes hat zur Folge, daß ein Satz von 89 APIs aus 13 Kategorien des Microsoft LAN-Managers verfügbar sein wird. Die LAN-Manager APIs bieten für die Entwickler von Applikationen leistungsfähige Funktionen: So kann z.B. die Interprozeß-Kommunikation zwischen Programmen im gesamten Netzwerk erfolgen. Damit ist die Entwicklung von verteilten Applikationen möglich. Außerdem sind Ressourcen-Zugriffsanfragen nach Servern oder Druckern sowie Netzwerk-Verwaltungsfunktionen, wie ferngesteuerte Operationen und die Überwachung von wichtigen Betriebszuständen auf dem Netzwerk verfügbar.

Mehr als 100 unabhängige Software-Hersteller und -Entwickler erarbeiten derzeit Applikationen, die LAN-Manager APIs nutzen. Die Implementierung eines Standardsatzes von APIs in OS/2 1.2 und im Microsoft LAN-Manager bedeutet, daß Entwickler von Netzwerk-Applikationen ihre Software nur gemäß einem einzigen Satz von Interface-Spezifikationen auslegen brauchen, um diese Software in IBM LAN Server- und Microsoft LAN-Manager Netzwerken einzusetzen.

Die IBM-Ankündigung fand in der gesamten Software-Branche eine breite Resonanz. Unternehmen wie Ashton-Tate, Revelation Technology, Saros Corporation und andere begrüßten die einheitlichen API-Spezifikationen und kündigten für ihre Produkte deren Unterstützung an. IBM selbst erwarb eine Lizenz für die Microsoft/3Com LAN-Manager NDIS. Mehr als 35 Hersteller haben bisher die NDIS unterzeichnet. Darunter Firmen wie AT&T, NCR, Hewlett-Packard, 3Com, Ungermann-Bass, Proteon, Interlan, Standard Microsystems, Intel, Western Digital, Retix, FTP Software, Madge Computers Limited und Sytek Inc. Die Ausrichtung der Version 1.2 des Betriebssystems OS/2 auf die LAN-Manager Schnittstellen-Spezifikationen (NDIS) gewährleistet, daß SNA-, NetBIOS- und Programme, die entsprechend dem IEEE 802.2-Programmier-Interface geschrieben worden sind, auch auf Ethernet-LANs laufen können. Die Verbindungsmöglichkeiten von Systemen unterschiedlicher Hersteller werden damit erheblich verbessert.

Microsoft gründet Multimedia Division

Die Microsoft Corporation gab in diesen Tagen die Gründung einer Multimedia Division bekannt, die sich ausschließlich der Entwicklung und Vermarktung von Multimedia Software und Endanwender-Produkten widmen wird.

Christian Wedell, Geschäftsführer der Microsoft GmbH, München-Unterschleißheim, und verantwortlich für das Geschäft des führenden Software-Anbieters in der Bundesrepublik, Österreich, der Schweiz und Osteuropa: »Innerhalb der nächsten Jahre muß man mit einem grundlegend neuen PC-Typus rechnen, der Hochgeschwindigkeitsprozessoren, hochauflösende Bildschirme und optische Medien verbindet. Dieser Multimedia-PC wird sowohl den privaten wie auch den beruflichen Anwender ansprechen. Die neue Multimedia Division von Microsoft soll unsere Arbeit in diesem Bereich forcieren und unterstützen.«

Geleitet wird die neue Division, die in zwei Gruppen unterteilt ist, von Min Yee, Vizepräsident von Microsoft. Geschäftsführer der »Multimedia System

Group«, zuständig für Multimedia Software-Entwicklung und -Vermarktung, ist Rob Glaser, seit kurzem Marketing Director des Geschäftsbereichs Netzwerke. Der ehemals zu Microsoft Press gehörige CD-ROM Produktbereich wird umbenannt in »Multimedia Publishing Group« und unter Leitung von Susann Lammers verantwortlich sein für die Entwicklung und Vermarktung von Multimedia Endanwender-Produkten, einschließlich der Programmer's Library und Bookshelf.



Microsoft Mäuse überspringen 2-Millionen-Hürde

Eine neue Rekordmarke notierten in diesen Tagen die Microsoft Mäuse: In den vergangenen 12 Monaten wurden über eine Million dieser beliebten Eingabehilfe, die vielfach als das »Original« bezeichnet wird, verkauft. Damit kletterte die Gesamtzahl der im Markt befindlichen Microsoft Mäuse auf über zwei Millionen Exemplare. Ein Ergebnis, das selbst Microsoft überraschte, da man vor Jahresfrist das Erreichen dieser Rekordmarke erst gegen Ende 1989 bzw. Anfang 1990 prognostizierte. Hauptanteil mit über 1,5 Mio. Stück an diesem Erfolg hat die im August 1987 eingeführte neue Microsoft Maus, die nicht nur durch ihr inzwischen vielfach preisgekröntes Design überzeugt, sondern vor allem auch durch Verschleißfreiheit und anwenderfreundliche Bedienbarkeit.

Die Microsoft Maus unterstützt den Anwender bei der Benutzung einer leistungsfähigen und komfortablen grafischen Bedieneroberfläche. Der Verkaufserfolg der Microsoft bezeugt denn auch die Durchsetzung grafischer Bedieneroberflächen. Weltweit werden heute bereits monatlich knapp 100.000 Packungen des Standards Microsoft Windows verkauft. Nicht mitgerechnet sind hier die vielen Tausend Windows-Versionen, die Hardware-Hersteller, wie zum Beispiel Schneider, Compaq und Siemens, direkt mit ihren Maschinen ausliefern.

Netzwerktreiber-Interface-Spezifikation

Die Netzwerktreiber-Interface-Spezifikation (NDIS) ist für Software-Entwickler ein Standard, auf dessen Basis sie Treiber für Netzwerk-Adapter und Kommunikations-Protokolle entwickeln können, die mit Microsoft LAN-Manager Software arbeiten. Darüber hinaus bietet die NDIS die Grundlage für den Protokoll-Manager, einen Konfigurations- und Anschlußtreiber. Microsoft und 3Com haben die NDIS für MS-DOS und MS OS/2 erstmals im April 1988 veröffentlicht. Die derzeitige Version ist seit Mai 1988 erhältlich.

Die modulare NDIS-Architektur minimiert den Entwicklungs-Aufwand für flexibel einsetzbare Netzwerk-Adapter und -Protokolle; Installations-Abläufe werden vereinfacht.

Treiber, die gemäß den in der NDIS definierten Schnittstellen geschrieben sind, arbeiten auch in Systemen mit anderen NDIS-Netzwerk- und Protokoll-Treibern. In der NDIS sind drei Klassen von Treibern spezifiziert, die die Unterstützung von verschiedenen Netzwerk-Adaptern und Protokollen vereinfachen:

- NDIS-Netzwerk-Adaptertreiber mit Low-Level-Zugriff auf Netzwerk-Adapter
- Protokoll-Treiber, die einen gesamten Ablauf, Transport, Netzwerk und »Data Link Protocol Layer« umfassen, um einen Kommunikationsdienst auf höherer Ebene zu unterstützen.
- Protokoll-Manager als ein spezieller Treiber, der einen Standard für die Lösung von Konfigurationsproblemen bildet und verschiedene Netzwerk-Adapter sowie Protokoll-Treiber einbinden kann.

Seit der Veröffentlichung der derzeit gültigen NDIS-Version vom Mai 1988, wird dieser Standard von der Netzwerk-Branche auf breiter Basis unterstützt. Zur Zeit sind mehr als 30 NDIS-Treiber am Markt verfügbar. Weitere 117 Treiber sind in der Entwicklung bzw. im Konformitätstest. Diese Treiber wurden für Netzwerk-Adapter von 3COM, AT&T, EXCELAN, Gateway Communications, Hewlett-Packard, IBM, Intel, INTERLAN, NCR, Proteon, Standard Microsystems, Sytek, Ungermann-Bass, Western Digital und anderen Herstellern erstellt.

Microsoft liefert zur Zeit das Netzwerktreiber-Entwicklungs-Kit (NDDK) zur Unterstützung der Entwickler von NDIS-Treibern aus. Das NDDK umfaßt alle Entwicklungswerkzeuge, die zum Erstellen von Treibern notwendig sind. Ferner beeinhaltet das NDDK Muster-Quellcode für IBM-Token-Ring und Intel-PC-586-Ethernet-Adaptertreiber. Microsoft entwickelt darüber hinaus eine Netzwerktreiber-Programmbibliothek, um Treiber-Software auf breiter Basis zu erstellen. Die Bibliothek wird in den USA auf einer Vielzahl von »Electronic Bulletin Boards« verfügbar sein, so daß auch Endanwender NDIS-kompatible Netzwerk-Adapterkarten-Treiber kopieren und laden können.

Alle Treiber der Microsoft LAN-Manager Netzwerktreiber-Bibliotheken sind von einem unabhängigen NDIS-Testlabor auf

Konformität geprüft worden. Sie werden immer paarweise getestet: sowohl für die MS-DOS- als auch für die MS OS/2-Umgebung. Beide Treiber sind jeweils für dieselbe Netzwerk-Adapterkarte ausgelegt.

Microsoft vergibt erste OEM-Lizenz für ROM-residentes MS-DOS an Emerson Radio

Microsoft gab vor wenigen Tagen die Lizenzvergabe für eine spezielle ROM-residente MS-DOS Version für die von Emerson Radio Corp. kürzlich angekündigten Lowcost-PCs bekannt. Damit ist Emerson der erste PC-Hersteller, an den Microsoft dieses 1981 entwickelte Industriestandard-Betriebssystem in ROM-residenter Version lizensiert. Die Lizenzvergabe an Hardware-Hersteller wie Emerson, die die Vorteile dieses Standard-Betriebssystems erkennen, eröffnet Microsoft neue Marktsegmente.

Bei »ROM-residenter« Installation wird das Betriebssystem direkt aus dem ROM (Read-Only-Memory) geladen und nicht über den »Umweg« von Diskette oder Festplatte gebootet. Dieses Verfahren bringt dem Anwender eine Reihe von Vorteilen:

- Unmittelbar nach dem Einschalten des Computers erscheint MS-DOS. Die üblichen Wartezeiten, die sich beim Laden des Betriebssystems ergeben, entfallen.
- Da das Betriebssystem bereits fest im Computersystem integriert ist, entfällt die nachträgliche Installation.

Das ROM-residente MS-DOS ist ein vollständiges Betriebssystem, das auf MS-DOS Version 3.3 basiert. Zusätzlich werden Festplatten-Partitionen über 32 Mbyte unterstützt. Die Emerson Radio Corp. wird das System mit ihren Modellen 800EC, 828 EC und 83836EC ausliefern.

Microsoft und NCR erweitern Vertrag über Workgroup-Produkte

NCR gab die Erweiterung der Lizenzen für Microsofts Workgroup-Produkte auf den Microsoft SQL-Server und den Microsoft Communications Server bekannt. Damit ist NCR, als einer der weltweit führenden Anbieter von PCs, der erste Hardware-Hersteller, der die gesamte Palette an Microsoft Workgroup-Produkten lizensiert.

»Mit unserer Adaption der Client/Server-Architektur sind wir nun auf dem Markt die ersten mit einer State-of-the-art-Netzwerktechnologie«, betont Tom Mays, Vizepräsident und General Manager der NCR Personal Computing Division. »Die langfristige strategische Allianz mit Microsoft ermöglicht es uns, sehr früh mit neuen Produkten und Verbesserungen auf den Markt zu kommen.«

Christian Wedell, Geschäftsführer der Microsoft GmbH erklärte hierzu gegenüber der Presse: »Für Microsoft ist die Lizenzvereinbarung eine Bestätigung unserer Unternehmensstrategie, die einen umfassenden Service für MS-DOS und MS OS/2-Workgroups vorsieht und NCR jetzt die Möglichkeit eröffnet, Komplettlösungen anzubieten.«

Die Microsoft OS/2 Workgroup-Produktpalette setzt sich zusammen aus:

- Microsoft LAN-Manager. Das erweiterte LAN-Betriebssystem läuft unter MS OS/2 auf dem Server und unterstützt MS-DOS, Microsoft Windows und MS OS/2-Workstations. Der Microsoft LAN-Manager bietet umfangreiche APIs (Application Programming Interface) zur Entwicklung verteilter Applikationen, fortschrittliche Fehlerschutz-Möglichkeiten sowie leistungsfähige Netzwerk-Verwaltungs-Werkzeuge. Der LAN-Manager wurde von führenden Systemanbietern, Netzwerkherstellern und Softwareentwicklern als Standard-Netzwerk-Plattform für PC-LANs angenommen.
- Microsoft SQL-Server. Ein Hochleistungs-Datenbank-Server für lokale Netzwerke. Er bietet eine Basis für transaktionsorientierte Applikationen, Tabellenkalkulationen u.a., die bisher auf Großcomputern implementiert sind. Der Microsoft SQL-Server arbeitet auf MS OS/2 Servern und besitzt Fähigkeiten, die mit Minicomputern und Mainfraim RDBMs vergleichbar sind. Applikationen wie dBASE IV, Lotus 1-2-3, Microsoft Excel und Paradox werden unterstützt.
- Microsoft Communications Server. Er erweitert die Anschlußmöglichkeiten von PCs und LANs hin zu Wide Area Networks und bietet den Benutzern einen flexiblen Zugriff auf IBM SNA-Netzwerke. Der Microsoft Communications Server bietet 3270-Terminal- und Druckeremulation sowie Datentransfer, APPC (advanced program to programm communication), sowie weitere Standard-Programmierschnittstellen und asynchrone Terminal-Emulation und Datentransfer.

Microsoft baut Marktführerschaft weiter aus

Ihre führende Position am weltweiten PC-Softwaremarkt weiter ausbauen konnte die Microsoft Corporation, Redmond/USA, die im zurückliegenden Geschäftsjahr (zum 30.6.) ihren Umsatz von 590,8 Mio US-Dollar auf 803,5 Mio US-Dollar (+ 36%) ausbaute. Im gleichen Zeitraum wuchs der Gewinn (nach Steuern) um 39% von 123,9 Mio US-Dollar auf 170,5 Mio US-Dollar.

Noch rasanter als die »Mutter« wuchs in den letzten 12 Monaten die deutsche Vertriebsgesellschaft, die Microsoft GmbH, Unterschleißheim-München, deren Umsatz von 92 Mio DM auf 140 Mio DM (+ 53%) klet-

terte. Hauptumsatzträger der bundesdeutschen Vertriebsgesellschaft sind neben den Standardbetriebssystemen wie MS-DOS und MS OS/2 die von Microsoft entwickelten zahlreichen professionellen Programmiersprachen sowie Applikationssoftware. Standardanwendungen wie das Tabellenkalkulationsprogramm Multiplan, das Zahlenmanagementprogramm Excel und das Textverarbeitungsprogramm Word halten heute bereits einen Anteil von 65% am GmbH-Umsatz. Mit Word bietet Microsoft das meistverkaufte deutschsprachige PC-Programm überhaupt an. Mit den Programmen Multiplan und dem neuen Excel für die grafische Bedieneroberfläche Windows konnte Microsoft im vergangenen Jahr in der Bundesrepublik auch im hart umkämpften Tabellenkalkulationsmarkt die Spitze einnehmen.

Christian Wedell, Geschäftsführer der Microsoft GmbH, erwartet im laufenden Geschäftsjahr weitere Wachstumsimpulse, die vor allem auch von dem Managementprogramm Excel für den neuen Oberflächenstandard Windows ausgehen werden. Schon heute verkauft Microsoft weltweit monatlich knapp 100.000 Windows-»Pakete«.

Umfang des Verfahrens Apple gegen Microsoft stark eingegrenzt

Im Verfahren Apple gegen Microsoft und Hewlett Packard hat der US-Bundesrichter W.W. Schwarzer eine Entscheidung getroffen, die den Umfang des Verfahrens erheblich einschränkt.

William H. Neukom, Vizepräsident und zuständig für die rechtlichen und unternehmerischen Belange der Microsoft Corp.: »Wir sind über diesen Richterspruch äußerst erfreut. Von 189 behaupteten Copyright-Verletzungen sind nach Auffassung des Gerichts 179 Punkte von dem zwischen Apple und Microsoft 1985 geschlossenen Vertrag gedeckt. Sie sind nicht mehr Prozeßgegenstand. Die verbliebenen 10 Punkte beziehen sich ausschließlich auf die Benutzung überlappender Hauptapplikationsfenster und gewisser Charakteristika von Sinnbildern in Windows 2.03. Die jetzt vorliegende Eingrenzung des Verfahrens bringt uns einer Lösung des Problems einen großen Schritt näher.«

In der jüngsten Entscheidung wurden folgende Feststellungen getroffen:

1. Die Verwendung von visuellen Darstellungselementen aus Windows 1.0 und den genannten Applikationen in Windows 2.03 wird von dem Lizenzabkommen zwischen Apple und Microsoft abgedeckt.
2. Die in Windows 2.03 verwendeten visuellen Darstellungselemente sind in Windows 1.0 und den genannten Applikationen enthalten, mit Ausnahme derjenigen, die sich auf die Benutzung von überlappenden Hauptapplikationsfenstern beziehen – im Gegensatz zu gekachelten Hauptapplikationsfenstern – und mit Ausnahme von spezifischen Änderungen im Erscheinungsbild und der Handhabung von Ikonen.
3. Microsoft und ihr Lizenznehmer Hewlett Packard haben ein Anrecht auf ein Teilurteil über Apples Klage auf Rechtsverletzung insoweit die Klage auf der Verwendung von visuellen Darstellungselementen aus Windows 1.0 und den genannten Applikationen in Windows 2.03 und NewWave basiert.

Microsoft geht davon aus, daß sich das Gericht und die Parteien in der nächsten Prozeßphase damit befassen werden, ob die in diesem Fall noch zu behandelnden visuellen Darstellungselemente originale oder schützbare Elemente oder in der Lizenzvereinbarung von 1985 beinhaltet sind.

In der Klage, die am 17. März 1988 im US District Court for the Northern District of California eingereicht wurde, vertritt Apple die Ansicht, Microsoft habe in Microsoft Windows 2.03 audiovisuelle Urheberrechte von Apple verletzt. Microsoft dagegen verweist auf das Lizenzabkommen von 1985 und betont, keine schützbaren Interessen von Apple verletzt zu haben.

Microsoft liefert neue Version 5.0 des schnellen PC Fortran Compilers aus – kompatibel zur VAX- und IBM-Syntax

Microsoft hat in diesen Tagen mit der Auslieferung von Microsoft Fortran, Version 5.0, begonnen. Der optimierende Compiler unterstützt die Syntax nach ANSI 77, VAX und IBM-VS. Die neue Version des weit verbreiteten Microsoft Compilers ist speziell für die Integration von PC-Arbeitsplätzen mit Mini- und Großcomputern ausgelegt, wie sie im Bereich der Wissenschaft und Technik zum Einsatz kommt. Mit Hilfe von Microsoft Fortran 5.0 können Programmierer die effizientesten und schnellsten Fortran-Programme schreiben, die für MS-DOS-, PC-DOS- oder MS OS/2-Entwicklungsumgebungen erstellbar sind. Sowohl die 16-Mbyte-Adressierfähigkeit als auch die dynamisch verbundenen Bibliotheken (Dynamic Link Libraries) unter MS OS/2 lassen sich unter Microsoft Fortran 5.0 nutzen.

Diese Funktionen zeichnen Microsoft Fortran nicht nur als produktivsten Fortran-Compiler auf dem PC-Markt aus, sie schaffen auch die Voraussetzung dafür, große komplexe Applikationen, wie sie typisch für den technisch-wissenschaftlichen Bereich sind, zu erstellen und zu nutzen. Der Anwender kann bereits existierende VAX-Programme auf seinen PC laden, so daß eine hohe Leistung ohne Begrenzung durch teure Hardware-Ressourcen zur Verfügung steht.

Die enorme Geschwindigkeit von PCs mit 80386-Mikroprozessoren ist erforderlich, um hochentwickelte Applikationen im Ingenieurwesen, in der Wissenschaft oder Mathematik auszuführen. Microsoft Fortran unterstützt komplexe Berechnungen und steigert dadurch in erheblichem Maße die Programmier-Produktivität. Außerdem lassen sich in Microsoft Fortran entwickelte Applikationen einfach auf andere Systemumgebungen portieren. Nahezu die gesamte VAX-Fortran-Syntax wird durch Microsoft Fortran, Version 5.0, unterstützt, so daß Großcomputer- und Minicomputer-Programme ohne großen Lernaufwand für den Software-Entwickler auf PCs portiert werden können. Microsoft geht davon aus, daß durch Fortran 5.0, unter MS OS/2 betrieben, rund 95 Prozent aller heute existierenden Fortran-Minicomputer-Applikationen abgedeckt werden. Die neue Compiler-Version ist GSA-geprüft und entspricht dem ANSI-77-Standard.

Microsoft Fortran 5.0 bietet alle Hilfen und Bibliotheken, die auch in den anderen Produkten der Microsoft Sprachenfamilie zur Verfügung stehen, einschließlich Grafik-Bibliothek, Microsoft CodeView Source-level Debugger, Microsoft Editor, LINK, NMake, LIB, MS OS/2 Unterstützung und erweiterte Syntax. Die Grafik-Bibliothek stellt Grundelemente für die Erstellung von Präsentationsgrafiken bereit. Sie ist ein neues Element in Microsoft Fortran, das alle notwendigen Bausteine zur Erzeugung von anschaulichen Darstellungen aus komplexen Zahlenreihen enthält. Microsoft Fortran unterstützt durch Schrift-, Farb- und Füllmuster die Gestaltung von Geschäfts-, Technik- und wissenschaftlichen Grafiken.

Mit der neuen Version des CodeView-Debuggers für Mehrfach-Anwendungen unter MS OS/2 sind Einzelschritt-Darstellung auf Programm-Ebene, schrittweise Protokollierung, Unterbrechungspunkte und die Darstellung des Inhalts eines lokalen Programm-Zwischenspeichers möglich. Der CodeView-Debugger kann darüber hinaus für dynamisch gekoppelte Bibliotheken und sehr große MS OS/2-Programme, bis zu 128 Mbyte, eingesetzt werden. Durch den Inkremental-Linker, der unter MS OS/2 und MS-DOS gleichermaßen arbeitet, wird der Zeitaufwand für das Verbinden von Dateien erheblich reduziert, da nur solche Dateien neu verbunden werden, in denen Änderungen durchgeführt wurden.

Der Microsoft Fortran-Compiler, Version 5.0, läuft auf IBM-PCs und kompatiblen Computern mit mindestens 320 Kbyte Arbeitsspeicher, MS-DOS 3.0 oder MS OS/2 1.1 (und höhere Versionen), die mit zwei doppelseitigen Disketten-Laufwerken oder einem doppelseitigen Disketten-Laufwerk und einer Festplatte ausgerüstet sind. Microsoft empfiehlt den Einsatz einer Festplatte.

Die Version 5.0 des Microsoft Fortran-Compilers kostet DM 1.095,– (exkl. MwSt.). Der Update-Preis von der Vorgängerversion auf Fortran 5.0 liegt bei DM 220,– (exkl. MwSt.).

Microsoft liefert QuickC 2.0 aus

Microsoft gab in diesen Tagen die Auslieferung des QuickC-Compilers 2.0 bekannt. Die neue Version macht das Programmieren in C einfacher und bringt für C-Programmierer, die eine integrierte Programmierumgebung bevorzugen, erhebliche Produktivitätsgewinne.

Programmierer werden durch den Microsoft QuickC-Ratgeber, Kurz-/Langmenüs, eine neue Dokumentation und das Computer Based Training (CBT) in die Lage versetzt, die leistungsstarke C-Programmiersprache schnell zu beherrschen.

Der Microsoft QuickC-Ratgeber ist ein neues Online-Referenz-System, das mit Hilfe der Hypertext-Technik auf Tastendruck eine Vielzahl von Informationen und Beispielprogrammen bereitstellt. Die Kurz-/Langmenüs helfen vor allem Einsteigern, sich schnell in der QuickC Programmierumgebung zurechtzufinden und produktiv Programme zu schreiben. Demselben Ziel dient das CBT. Ein weiteres Element von Microsoft QuickC 2.0 ist »C – Programmieren leichtgemacht«, ein Lehrbuch zur Vermittlung der Grundlagen der Programmiersprache C und der spezifischen QuickC-Compiler Funktionen.

QuickC 2.0 bietet nun inkrementelles Kompilieren und Linken, das zu einer erheblichen Beschleunigung der Kompiliergeschwindigkeit führt. Die Folge: Die Version 2.0 ist zwei- bis dreimal schneller als die bisherige Version. Außerdem enthält der neue Microsoft QuickC-Compiler einen Inline-Assembler. Mit ihm können Anwender editieren, kompilieren/assemblieren sowie per Debugger-Unterstützung Fehler suchen, ohne die Programmierumgebung zu verlassen.

Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, zu den Vorteilen des neuen Microsoft QuickC-Compilers: »Jeder bisherige Anwender eines QuickC-Compilers sollte wegen der erheblichen Produktivitätsverbesserungen auf die neue Version 2.0 umsteigen. Programmierer, die bisher noch nicht in C programmiert haben, weil sie der Meinung waren, dies sei zu schwierig, werden überrascht sein, wie einfach und schnell die Programmierung mit QuickC 2.0 geht.«

Eine weitere neue Funktion im Microsoft QuickC-Compiler 2.0 ist die Unterstützung aller Speichermodelle, wie Small, Medium, Compact, Large und Huge. Außerdem werden die Schlüsselworte near, far und huge unterstützt, so daß der Anwender Speichermodelle im Hinblick auf eine maximale Effizienz mischen kann.

Die Systemvoraussetzungen für die Nutzung des Microsoft

QuickC-Compilers Version 2.0 sind: IBM-PC oder kompatibler Computer mit mindestens 512 Kbyte Arbeitsspeicher, DOS 2.1 bzw. höhere Version und entweder zwei doppelseitige 5,25-Zoll-Laufwerke oder ein doppelseitiges 3,5-Zoll-Laufwerk. Microsoft empfiehlt nachdrücklich den Einsatz einer Festplatte. Der QuickC-Compiler 2.0 unterstützt CGA-, VGA-, EGA- und Hercules-Grafikkarten.

Die Entwicklung des Microsoft QuickC-Compilers 2.0

Programmiersprachen waren bei der Gründung von Microsoft die Basis der Unternehmensentwicklung. Sie bilden auch heute noch einen Schwerpunkt in der Produktpalette und haben den Fortschritt in der PC-Branche vermutlich stärker beeinflußt als jede andere Software. Microsoft Programmiersprachen wurden und werden auf nahezu allen führenden Applikationen, wie MS-DOS, Lotus 1-2-3 oder WordPerfect, eingesetzt und sind auch für die Software-Plattform der Zukunft, MS OS/2-Presentation Manager, verfügbar.

Microsoft wird häufig gefragt, wie darüber entschieden wird, welche Richtung die Entwicklung der Produkte nehmen soll. Die folgenden Hintergrundinformationen sollen einen Einblick in die Entstehung eines Microsoft Produkts geben, wobei es in diesem Fall um die Entwicklung, die Erstellung und das Testen des Microsoft QuickC-Compilers 2.0 geht.

Der Microsoft QuickC-Compiler 1.0

Mit dem Microsoft QuickC-Compiler 1.0 stieg das Unternehmen im Oktober 1987 in das C-Compiler-Marktsegment ein. Die Entwicklung für dieses Produkt begann bereits 1986 Formen anzunehmen und folgte quasi dem Erfolg von QuickBASIC 2.0. Basierend auf der Resonanz von Microsoft QuickBASIC und frühzeitigen Marktforschungen, zeigte sich ein großer Markt für einen C-Compiler, der die vollständige Implementierung von Microsoft C, zusammen mit einer interaktiven, integrierten Programmierumgebung, darstellt. Dazu sollte der Verkaufspreis attraktiv genug sein, so daß auch nichtprofessionelle Programmierer für einen Versuch, die leistungsfähige Programmiersprache C auszuprobieren, zu gewinnen sind.

Die Untersuchungsphase

Das vierte Quartal 1987 brachte für die Microsoft-Programmiersprachen umwälzende Änderungen: Microsoft stellte mit Microsoft QuickBASIC 4.0, Microsoft C 5.0 und Microsoft QuickC-Compiler 1.0 drei wichtige Sprachenprodukte vor und begann mit der Auslieferung.

Microsoft QuickBASIC 4.0 bot in Form des P-Code-Interpreters eine neue Programmiersprachen-Technik, bei der das interaktive Verhalten eines Interpreters mit der Geschwindigkeit eines Compilers verbunden wurde. Der optimierende Microsoft C-Compiler 5.0 bot dem professionellen C-Programmentwickler eine bisher nie gekannte Code-Optimierung. In QuickC 1.0 wurde die integrierte Programmierumgebung für C verfügbar.

Microsofts Philosophie ist es, die Produkte ständig den Anforderungen derzeitiger und zukünftiger Anwender anzupassen. Eine 1987 in Auftrag gegebene umfassende Studie über die Anwender der Microsoft Programmiersprachen sollte dabei die neuen Richtungen aufzeigen. Aus der Studie, die alle Programmiersprachen umfaßt, werden nachfolgend nur die Ergebnisse von QuickC dargestellt.

Da die erste Version von Microsoft QuickC zum Start der Untersuchung bereits ausgeliefert war, konnte Microsoft die Erfahrungen der Anwender während der ersten zwei Monate erkunden. Vorrangiges Ziel der Studie war es, Antworten auf folgende Fragen zu finden:

- Wer sind die Anwender?
- Welche Faktoren führten zum Kauf?
- Was veranlaßte die Käufer, QuickC aus dem gesamten Angebot auszuwählen?
- Wie und auf welchen Computern setzen die Anwender QuickC ein?

Die Untersuchung zeigte keine besondere Gewichtung hinsichtlich beruflicher Funktion, Sparte oder Branche. QuickC wurde von Systemanalytikern, Programmierern, Ärzten und Rechtsanwälten gekauft. Gemeinsam war diesen Anwendern lediglich, daß sie den PC als Werkzeug für ihre Arbeit nutzten. Wie die Untersuchung zeigte, verfügten die QuickC-Kunden über eine anspruchsvolle Hardware-Ausstattung: 58% besaßen Computer auf 80286- oder 80386-Prozessorbasis, 88% dieser Computer waren mit einem Arbeitsspeicher von mindestens 640 Kbyte ausgestattet und 70% der eingesetzten Betriebssysteme waren MS-DOS Version 3.2 oder höhere Versionen.

Außerdem bestand die Mehrheit der Anwender nicht aus Anfängern: 72% hatten mehr als fünf Jahre Programmier-Erfahrung. Viele Befragte waren allerdings Einsteiger in der Programmiersprache C. 49% der befragten QuickC-Anwender gaben an, daß der Grund für den Kauf von QuickC das Erlernen der Programmiersprache C ist.

Nach 60 Tagen (oder weniger) Arbeit mit QuickC definierten 72% der Anwender den primären Einsatzzweck als Entwicklungsinstrument von Software. 55% entwickelten Software für den Verkauf oder für den Einsatz bei anderen Anwendern, während 28% Programme schrieben, um zu lernen oder die Freizeit damit zu gestalten.

Für den typischen QuickC-Anwender sind eine Reihe von Produkt-Eigenschaften von höherer Bedeutung als diejenigen traditioneller Benchmarks, wie Kompilierzeit oder Ausführungsgeschwindigkeit. Die folgende Tabelle zeigt eine Aufstellung der verschiedenen Faktoren nach ihrer Gewichtung durch die Anwender.

(Anmerkung: Die Anwender wurden gebeten, diese Gewichtung auf einer Skala von 1 – 5 vorzunehmen, wobei 5 für »sehr wichtig« und 1 für »nicht wichtig« stand.)

Eigenschaft	Wertung
Gesamt-Gegenwert für das investierte Geld	4,51
Ansehen des Herstellers	4,25
Qualität der Dokumentation	4,21
Zuverlässigkeit	4,20
Integrierter Quellprogramm-Debugger	4,17
Einfache Anwendbarkeit	4,13
Preis	4,04
Vollständig integrierte Programmier-Umgebung	3,90
Kompatibilität mit Microsoft C 5.0	3,87
Kompilierunsgeschwindigkeit	3,80
Ausführungsgeschwindigkeit	3,78

Durch eine weiterführende Befragung stellte sich heraus, daß speziell für die Einsteiger in die C-Programmierung mit »einfacher Bedienbarkeit« die Zeitspanne verstanden wird, die investiert werden muß, um produktiv zu werden.

Auf der Basis dieser Informationen stellte sich Microsoft die Frage, was zu tun ist, um QuickC zu einer einfach erlernbaren C-Programmierumgebung für nahezu 50% der Anwender zu machen, die das Produkt kaufen, um C zu lernen. Gleichzeitig sollte jedoch die Leistung geboten werden, die für die professionelle Software-Entwicklung benötigt wird.

Die Lösung ist Microsoft QuickC 2.0, bei dem neue Standards gesetzt wurden, um den Anforderungen der Anwender gerecht zu werden.

Implementierung

Das QuickC 2.0 Team bestand aus Compiler-Experten, die Microsoft C 5.1 entwickelt haben, sowie aus Mitarbeitern des Microsoft QuickBASIC 4.0 Teams und aus Test- und Anwenderschulungs-Teams.

Bei der Definition der Spezifikationen für Microsoft QuickC 2.0 waren die Entwicklungs-, Anwenderschulungs-, Marketing- und Test-Teams beteiligt. Jede vorgeschlagene Funktion mußte folgendes Kriterium erfüllen: Trägt sie dazu bei, QuickC einfacher handhabbar zu machen, während gleichzeitig die Leistung vorhanden ist, um die Aufgaben zu erfüllen?

Die wichtigsten neuen Funktionen in QuickC 2.0 wurden quasi durch die Anwender entwickelt. Microsofts Herausforderung als Programmiersprachen-Entwickler bestand darin, mit technischem Wissen sowie durch Kreativität und Innovation ein Produkt zu erstellen, das für den Anwender wirklich nützlich ist. Die folgenden Ausführungen beziehen sich auf zwei der wichtigsten Elemente der neuen QuickC 2.0 Technik und darauf, wie diese Elemente zum Nutzen des Anwenders eingesetzt werden.

Der QuickC-Ratgeber

Der Microsoft QuickC-Ratgeber ist ein Online-Hilfesystem auf Hypertext-Basis. Hypertext versteht sich als eine bestimmte Art der Text-Präsentation, die es dem Anwender erlaubt, eine bessere Kontrolle über den jeweiligen Begriff, den Begriffs-Zusammenhang und die Menge der dargestellten Informationen zu haben. Produkte wie die Apple HyperCard, haben die Vorteile des Hypertext-Konzeptes demonstriert. Microsoft ist der Meinung, daß Hypertext, der jeweiligen Programmier-Umgebung angepaßt, einen wesentlichen Beitrag zur besseren Arbeit mit Microsoft QuickC 2.0 leistet.

Ein Hilfesystem in einer integrierten Programmier-Umgebung muß – um nützlich zu sein – umfassend Auskunft geben. Es muß mehr Informationen beinhalten, als üblicherweise in einem Handbuch zu finden sind. Außerdem soll die jeweilige Hilfe kontextbezogen sein, gleichgültig, an welcher Stelle der Programmierumgebung sich der Anwender gerade befindet. Vor und während der Entwicklung des QuickC-Ratgebers wurde ermittelt, wie Programmierer Handbücher benutzen. Dabei stellte sich heraus, daß Programmierer häufig zusätzliche Informationen und Programmbeispiele suchen, für die sich in Handbüchern meistens der Fußzeilenhinweis »Siehe auch...« findet. Im QuickC-Ratgeber sind diese »Siehe auch...«-Referenzen implizit und explizit vorhanden. Außerdem gibt es eine Reihe von Anklick-Punkten (Hot Spots) und Boxen. Sucht der Anwender beispielsweise irgendeine C-Bibliotheks-Funktion, so wird er verschiedene Boxen für Beschreibung, Details und Beispiele finden. Durch die Plazierung des Cursors auf die Beispielbox und Klicken der Maus erscheint unmittelbar ein Beispielprogramm für die jeweilige Bibliotheks-Funktion. Da viele Menschen über Beispiele am einfachsten eine Programmiersprache lernen, ermöglicht der QuickC-Ratgeber das Aufrufen eines Beispielprogramms in ein Fenster, die Kompilierung dieses Programms und den sofortigen Test. Auf der anderen Seite gibt es »Implizit-Verbindungen« (Implizit Links), die zu Hinweisen und Hilfen führen, unabhängig davon, an welcher Stelle sich der Anwender gerade befindet. So gibt es beispielsweise für jedes C-Schlüsselwort und jede Bibliotheksfunktion eine implizite Verbindung zur jeweiligen ursprünglichen Beschreibungsseite. Unabhängig davon, in welchem Programmteil sich der Anwender gerade befindet, erscheint die zugehörige Beschreibung auf dem Bildschirm, wenn er den Cursor auf eine Bibliotheksfunktion setzt und die Hilfe-Taste drückt. Auf diese Weise erhält der Anwender schnell und einfach die notwendigen Informationen.

Der Microsoft QuickC-Ratgeber ist vollständig unabhängig von der Systemumgebung implementiert. Er läßt sich deshalb auf einfache Weise in jedes andere Produkt einbauen und ist bereits bei Microsoft QuickBASIC 4.5 verfügbar. Die QuickC-Ratgeberdateien sind um bis zu 55% gegenüber dem ursprünglichen ASCII-Format komprimiert. Auf diese Weise war es möglich, nahezu 1.400

Dokumentationsseiten auf einer einzigen Diskette mit 360 Kbyte Kapazität unterzubringen. Insgesamt repräsentiert der QuickC-Ratgeber mehr als 1.800 Dokumentationsseiten.

Programmierumgebung der zweiten Generation
In Bezug auf den Microsoft QuickC-Compiler 2.0 gibt es eine Reihe weiterer wichtiger Punkte, bei denen Microsoft modernste Compiler-Technologie eingesetzt hat, damit der Anwender so schnell wie möglich von der Eingabe zu einem ablauffähigen Programm kommt. Durch die Untersuchung der Art und Weise, wie Anwender mit Microsoft-Produkten arbeiten, wurde deutlich, daß ein typischer Arbeitsgang aus einem Zyklus von Programmeingabe, Kompilierung, Fehlersuche, Umschreiben des Programms, Kompilierung, Fehlersuche usw. besteht. Mit QuickC 2.0 wurden deshalb Compiler- und Debugger-Subsysteme entwickelt, die eine bislang nicht vorstellbar kurze Zeit für das Durchlaufen eines solchen Arbeitszyklus ermöglichen. Konfigurierbare Eingabetasten sowie die »Eingabe und Weiter«-Möglichkeit bei der Fehlersuche sind weitere Funktionen, die den Anwender dabei unterstützen, schnell und einfach das Ziel seiner Bestrebungen, die Erstellung eines ablauffähigen Programms, zu erreichen.

Unterstützung für DDE-Entwicklungen unter MS-Windows

DDE (»Dynamic Data Exchange«) ist ein Protokoll zur Realisierung dynamischer Programmkommunikation unter MS-Windows. Es hat gute Aussichten, in Zukunft in jede Windows-Applikation integriert zu werden, so daß die Applikationen untereinander automatisch Daten austauschen und Befehle aufrufen können, wie dies beispielsweise heute schon mit Microsoft-Excel möglich ist.

Problematisch ist jedoch für den Windows-Entwickler die Integration des DDE-Protokolls in seine Applikation. Es sind eine Vielzahl von Features zu beachten und zu implementieren, etwa umfangreiche Datenorganisation, Synchronisationsprobleme, Zerlegung von DDE-Befehlssequenzen und Ausnahme- und Fehlerbehandlungen.

Deshalb hat die Firma Buchheit software research in Karlsruhe (Telefon 0721/366776) einen DDE-Treiber als dynamische Link-Library entwickelt, der auch von ihr vertrieben wird. Der Treiber vereinfacht mit einem DDE-API die Ansteuerung der DDE-Schnittstelle erheblich. Er wird über Funktionsaufrufe aus der Applikation aufgerufen und alle komplexen Vorgänge sind im Treibercode verborgen. Er kann sowohl auf der Client- als auch auf der Server-Seite von DDE verwendet werden. Insbesondere läßt sich mit dem Treiber eine einfache Einbindung von PC-Hardware-Erweiterungskarten in Windows-Software erreichen. Der Treiber kostet DM 900,- (+MWSt) und wird mit umfangreicher Dokumentation ausgeliefert.

Vom gleichen Hersteller ist ein Paket mit Namen »Integrator's DDE-Utilities« lieferbar. Hierbei handelt es sich um fertige Windows-Applikationen, die zur Simulation von DDE-Clients und -Servern eingesetzt werden können. Damit lassen sich DDE-Übertragungen (beispielsweise aus Excel heraus) leichter testen und simulieren. Eine weitere Applikation erlaubt die Analyse von Daten, die über DDE übertragen werden. Die DDE-Utilities sind insbesondere für Entwickler gedacht, die ihre Software durch Integration von Standardapplikationen entwickeln (etwa auf der Grundlage Excel und der Datenbank Omnis-Quartz) und hierzu auf die DDE-Kommunikation zurückgreifen. Die DDE-Utilities kosten DM 300,- (+MWSt).

Echtzeit-Kommunikation mit Ganymed

Die Kommunikations-Software Ganymed der Firma Quasar hat die Postzulassung für die seit kurzem verfügbaren schnellen Modems mit Fall-Back-Prozedur erhalten. Der Telebox-Dienst der Deutschen Bundespost benutzte Ganymed bereits während der CeBIT für Präsentationen und Messedemos. Durch die Zulassung für höhere Übertragungsgeschwindigkeiten wird Ganymed nun sowohl für Rechnerkopplungen, den Anschluß von Waagen- und Kassenperipherie etc. im Supermarkt, als auch wegen der automatisierbaren Abläufe und Zeitsteuerung für Filialsteuerungen, die vollständige Fernwartung und Ferndiagnose von UNIX, C-DOS, MS-DOS sowie PS/2-Systemen eingesetzt.

Die wesentlichen Vorteile: Ganymed ist eine vollständig deutsche Entwicklung und nutzt deshalb die Stärken der deutschen und europäischen Postinfrasturktur, die sonst den Anwendern verschlossen bleiben.

Weiterhin stelle Ganymed durch seine Flexibilität die besonderen Bedürfnisse der einzelnen Anwender sicher. Clou von Ganymed ist der Parallelbetrieb zwischen Kommunikation und PC-Funktionen, dadurch lassen sich übertragene Dateien und Informationen ohne manuellen Zwischenschritt direkt elektronisch weiterverarbeiten.

Turbo Backup jetzt noch schneller

Die neue Version Turbo Backup 5.0 soll alle Vorzüge einer blitzschnellen, komfortablen Datensicherung mit der Möglichkeit verbinden, Daten zwischen zwei PCs zu übertragen.

Turbo Backup gehört zu den

schnellsten Backup-Utilities und macht die Datensicherung durch einfache Bedienung zu einer wirklichen »Nebensache«.

Turbo Backup kann über Stapel- und Kommandodateien automatisiert werden, so daß die tägliche Datensicherung dann über einen einfachen Tastendruck erfolgt. Der Anwender erhält in einer leicht verständlichen Anleitung Hinweise darauf, in welchen Perioden er eine Vollsicherung bzw. bloße Ergängzungssicherungen vornehmen sollte, um jederzeit mit dem geringsten Aufwand eine optimale Sicherung zu erzielen.

Turbo Backup kann Programme vor der Sicherung auf Virus-Befall prüfen. Wird eine verdächtige Abweichung vom Originalzustand festgestellt, ist bei regelmäßiger Datensicherung selbst die Radikalkur »Formatieren der Festplatte« kein Problem mehr. Es besteht zu jeder Zeit eine gültige Diskettenserie zur Restaurierung.

Die neue Version 5.0 von Turbo Backup bietet eine ganz besondere Überraschung: Das Programmodul Turbo Wire ermöglicht auch eine Datensicherung bzw. Datenübertragung zwischen zwei PCs über die serielle Schnittstelle. Bei PCs mit unterschiedlichen Diskettenformaten ist dies eine erstaunlich einfach Lösung, sie wieder kompatibel zu machen. Wer die Daten seines Laptops auf den Tisch-Computer übertragen will, braucht beide Geräte nur über ein serielles Nullmodem-Kabel zu verbinden.

```
TURBO BACKUP Ver. 5.00   UTI-MACO SOFTWARE   (C) Dansk Data Support 1986

                        Verzeichnis wählen

C:\        DOS       SAFE-GRD.30   50TEST    PCF
OW         TEXTE     TN            3270      PS2
ALOCK      NU        FASTBACK      TB-ENG    VERTRIEB
FB515      TB50      LUCID

                     Unterverzeichnis [JA  ]

                     Dateien wählen *.*

Ergänzungssicherung [NEIN]          Innerhalb der Serie [NEIN]

                     SICHERUNG BEGINNEN

<ESC>: Zurück    <F1>: Hilfe    <>-<>: Funktion    <┘>: Bestätigen
```

Mit dem Menü von Turbo Wire läßt sich die Datenübertragung in beide Richtungen exakt steuern. Turbo Backup 5.0 läuft auch unter MS-DOS 4.0, ist netzwerkfähig und kostet DM 569.

Erster EISA-Industrie-Chipsatz

Intel präsentiert den industrieweit ersten Chipsatz zum Implementierung des 32-bit-EISA-Busses (Extended Industry Standard Architecture). Der EISA-Bus-Chipsatz 82350 von Intel besteht aus drei Komponenten für den Einbau auf der PC-Systemplatine sowie einem Bus-Master-Interface Controller für Einsteckkarten.

Muster des Chipsatzes sind seit knapp einem Monat verfügbar, womit Intel – laut Paul Otellini, Vice President und General Manager der Folsom Microcomputer Division von Intel – den agressiven Zeitplan einhielt, den sich das Unternehmen anläßlich der Ankündigung des EISA-Busses im September letzten Jahres setzte.

Otelline kommentierte: »EISA erweitert die Einsatzmöglichkeit für derzeitige 16-bit-ISA-Einsteckkarten und eröffnet gleichzeitig einen Weg zur Leistungssteigerung durch die Verwendung von Busmaster-Zusatzplatinen«. Der ISA-Bus ist auch unter der Bezeichnung PC-AT-Bus bekannt.

Otellini fügte hinzu, daß die 82350-Familie Intels 32-Bit-Bus-Programm abrunde und den Kunden die Wahl unter zwei Industriestandard-Optionen biete. »Wir haben bereits einen gut eingeführten Chipsatz – den 82311 – der mit der Micro Channel-Architektur kompatibel ist. Jetzt erwarten wir auch künftige Systeme auf der Basis unserer Familie 82350«.

Systemplatinen-Bauteile

Zur Familie 82350 zählen der ISP-Baustein (Integrated System Peripheral) 82357, der EISA-Bus-Controller (EBC) 82358 und drei EISA-Bus-Buffer (EBB) 82352.

Das ISP 82357 ist ein hochintegriertes Bauelement mit den Funktionen Taktgeber/Zähler, Interrupt-Steuerung, Bus-Arbitrierung, DRAM-Auffrischung und direkter Speicherzugriff (DMA) auf einem einzigen Chip. Durch den Einsatz von Intels 1,5-• m-Technologie CHMOS III erzielt das ISP eine DMA-Transferrate von 33 Mbyte/s – mehr als achtmal so viel, wie der PC-AT-Bus. Das ISP übernimmt auch die Bus-Arbitrierung und entscheidet damit, welche logische Funktion (CPU, Busmaster oder DMA) zu einer bestimmten Zeit Zugriff auf den Bus erhält.

Der EBC 82358 bildet die Schnittstelle zu drei verschiedenen PC-Bussen: dem 8-/16-bit-ISA-Bus, dem 32-bit-EISA-Bus und dem Bus der Host-CPU. Er übersetzt die Informationen zwischen den Einsteckkarten und stellt sicher, daß 8- und 16-bit-ISA-Slave- und -Master-Karten effizient mit EISA-Zusatzkarten kommunizieren können. Zusätzlich erkennt der EBC sowohl den 32-bit-386 als auch den i486-Mikroprozessor und arbeitet mit beiden zusammen.

Der Einbausatz für die Systemplatine enthält drei EBB 82352. Sie enthalten die Buffer-Logik für Adressen, Daten und Parität in jeder von drei Betriebsarten und ersetzen damit nicht weniger als 17 TTL-Bauteile. Darüber hinaus helfen die EBBs den Systementwicklern, die kritischen EISA-Timing-Anforderungen einzuhalten.

Busmaster-Element für Einsteckkarten

Intelligente Zusatzplatinen können in EISA-Systemen durch den Bus Master Interface Controller (BMIC) Intel 82355 eingesetzt werden. Der BMIC bietet eine flexible Schnittstelle zwischen den lokalen Funktionen der Busmaster-Platine und dem EISA-Busmaster-Protokoll.

Der BMIC 82355 arbitriert die Busherrschaft und kann die Steuerung des Busses von der Host-CPU übernehmen. Das entlastet die CPU von busintensiven Aufgaben in anspruchsvollen Anwendungen wie Datenbanksystemen und File-Servern. In Zusatzkarten für Lokalnetzwerke, Plattenlaufwerken und Grafik ist der BMIC 82355 ebenfalls von Nutzen.

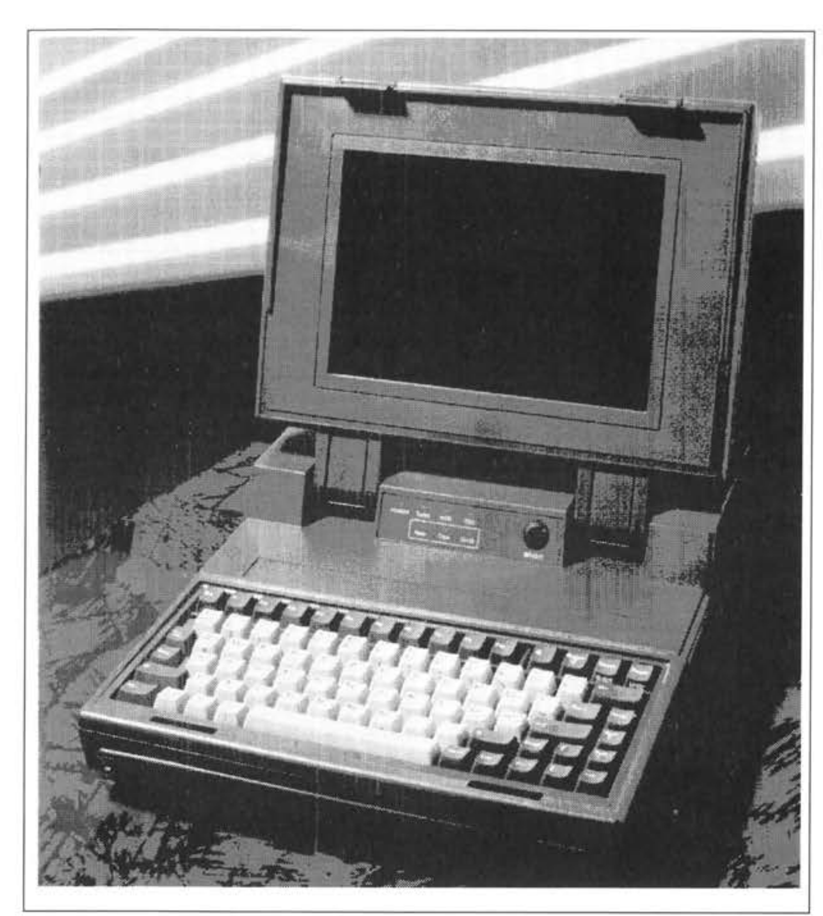
Neue Optionen für Compaq SLT/286

Compaq stellt zwei neue Optionen für den Compaq SLT/286 Laptop-Computer vor, das externe Batterie-Ladegerät und den Auto-Adapter. Das externe Batterie-Ladegerät wird an das externe Netzteil des Compaq SLT/286 angeschlossen und lädt einen leeren Batterie-Block in eineinhalb Stunden wieder auf. Das Ladegerät faßt zwei Batterie-Blöcke, die jeweils einzeln geladen werden. Die Ladezeit hängt von der im Batterie-Block verbliebenen Restkapazität ab.

Über den Auto-Adapter kann der Compaq SLT/286 Laptop an den Zigarettenanzünder eines Autos angeschlossen werden und die Autobatterie als externe Spannungsquelle nutzen.

Die beiden neuen Optionen für den Compaq SLT/286 sind über alle autorisierten Compaq-Fachhändler erhältlich. Die empfohlenen Verkaufspreise für das externe Batterie-Ladegerät und den Auto-Adapter liegen bei DM 329,- bzw. DM 159,-.

In gleichem Maße, wie die beiden neuesten Optionen die Einsatzmöglichkeiten des Compaq SLT/286 verbessern, erhöht die Basis-Erweiterungseinheit dessen Funktionalität. Die Basis-Erweiterungseinheit bietet Steckplätze für zwei 8-/16-Bit-Erweiterungsplatinen im Industrie-Standard und verfügt außerdem – wie der Compaq SLT/286 selbst – über eine parallele Schnittstelle sowie Anschlüsse für einen externen VGA-Monitor und eine externe Enhanced-Tastatur. Dank dieser Schnittstellen können Peripheriegeräte an die Basis-Erweiterungseinheit angeschlossen bleiben, wenn der Compaq SLT/286 »auf die Reise geht«. Die bereits im Oktober 1988 angekündigte Basis-Erweiterungseinheit wird derzeit weltweit an autorisierte Compaq-Fachhändler ausgeliefert.



Neuer Laptop vom Computer-Himmel

Das Münchner Systemhaus Computer Sky päsentiert den neuen Laptop »Sky Plasma 286« zu einem Preis von DM 6950. Der Sky Plasma 286 ist ein vollwertiger AT mit 286er CPU, 12 MHz Taktfrequenz (16 MHz Landmark), bis zu 8 Mbyte ausbaubarem RAM-Speicher, EMS-Standard und page-interleave. Der Bildschirm besteht aus einer Plasmaanzeige mit 640x400 Bildpunkten. Das stabile Gehäuse im Aktenkofferformat hat die Ausmaße 31x36x9cm.

Laptops erfreuen sich immer größerer Beleibtheit. Nicht nur für Vertreter in der Industrie sind tragbare Computer für die Präsentation von Daten unumgänglich. Der Laptop wird immer mehr die Reiseausführung des geliebten Heimcomputers. Dies spiegelt sich auch im Preis wieder. Laptops werden erschwinglich. Der Sky Plasma 286 AT ist mit 12 MHz ein sehr schneller Laptop. Der Arbeitsspeicher beträgt 1 Mbyte und ist bis zu 8 Mbyte auf der Platine ausbaubar. Zudem ist optional ein 80287 Coprozessor integrierbar, so daß der Laptop für anspruchsvolle Aufgaben in der Bearbeitung großer Datenmengen geeignet ist.

Der Datentransfer ist mit Hilfe eines 3,5-Zoll-Laufwerks, sowie eines seriellen und eines parallelen Ports möglich. Zusätzlich lassen sich ein 5,25-Zoll-Laufwerk und ein externer CGA-Monitor anschließen. Die Eingabe erfolgt über eine Tastatur mit 81 Tasten mit integriertem numerischem Teil. Der Gasplasma-Bildschirm umfaßt 80 Buchstaben mal 25 Zeilen. Einschließlich der eingebauten 150-Watt-Stromversorgung wiegt der AT im Koffer 7 Kg.

ScanMan Plus

Die Münchner Logi GmbH, Tochter der Schweizer Logitech International S.A., stellt das neueste Produkt aus der ScanMan-Familie vor, den ScanMan Plus. Den Handscanner gibt es in zwei Versionen: für IBM-PC, XT, AT, PS/2 (Modell 25 und 30) sowie kompatiblen Systemen und für IBM PS/2 Modell 50 und höher. Der ScanMan Plus ist in der englischen Version ab Juli, in der deutschen ab September für voraussichtlich DM 740 lieferbar.

Der ScanMan Plus enthält neben dem 400g leichten Scanner eine Steckkarte, die beiden verbesserten Software-Programme Paint-Show Plus (Version 2.2) und ScanMate (Version 1.2) sowie zwei Handbücher. Mit einer Abtastbreite von 106 mm und Auflösungen von 100, 200, 300 und verbesserten 400 Dpi erweist sich der ScanMan Plus auch für den professionellen Einsatz gerüstet.

Auffällig beim ScanMan Plus ist das neue Design und eine verbesserte Software. Mit einer Breite von 13,5 cm, einer Länge von 14 cm und einer Höhe von 3,5 cm ist der Scanner größer als sein Vorgängermodell. Damit liegt er nicht nur besser in der Hand, sondern ermöglicht auch eine ruhigere und damit exaktere Führung beim Scannen.

Drei Gummiwalzen sorgen dafür, daß der ScanMan Plus leichter über die Vorlage läuft und Verzerrungen beim Abscannen ausgeschaltet werden.

Alle wichtigen Einstellmöglichkeiten, wie Helligkeit und Wahl der Scan-Kontraste sind bedienerfreundlich an der linken Seite angebracht und können problemlos beim Scannen verändert werden. Neu ist auch das gelb-grüne Licht des ScanMan Plus. Damit können jetzt auch Vorlagen mit roten Farbanteilen gescannt werden.

Weiterentwickelt wurde auch ScanMate, die eigens für den ScanMan entwickelte Software von Logitech. Mit DOSScan wurde zudem ein nützliches Werkzeug in die Software integriert. DOSScan zeigt dem Anwender bereits während des Abscannens das Bild auf dem Bildschirm. Damit kann schon während des Scann-Vorgangs mittels der realtime display-Funktion die Qualität der Abbildung kontrolliert werden.

Mit Microsoft-Seminaren sicher in die Zukunft

Die ständige Leistungssteigerung in der Hard- und Software-Entwicklung läßt den Fachkräften der Industrie kaum eine Chance, die Möglichkeiten dieser Technologie in gleicher Schnelligkeit umzusetzen. Dabei ist es von sehr großer Bedeutung, Arbeitsbereiche mit Hilfe der EDV zu rationalisieren. Um die Integration der EDV und damit eine Arbeitserleichterung im betriebswirtschaftlichen Alltag zu erreichen, setzt Microsoft die Institut-Seminare in München und Düsseldorf fort.

Diese Spezialseminare des Microsoft Instituts vermitteln in kleinen Gruppen intensiv all das, was zum Einstieg in die Programmentwicklung nötig ist. Modernste Trainingsmethoden sowie PC-Demonstrationen und -Übungen sind selbstverständlich. Die Dozenten befassen sich auch im persönlichen Gespräch ausführlich mit den individuellen Forderungen und Problemen der Teilnehmer. So bekommen professionelle Entwickler durch professionelle Schulung die Möglichkeit, ihren hohen Wissenstand den neuen Gegebenheiten anzupassen. Das Microsoft Institut bietet Seminare in drei Bereichen an:

Systemsoftware

Diese Seminare sind für PC-Software-Entwickler bestimmt:
- Microsoft OS/2
- OS/2-Presentation Manager
- Microsoft Windows
- Microsoft LAN-Manager
- Microsoft SQL-Server

Applikationen

Diese Seminare sind in erster Linie für die Mitarbeiter des EDV-Benutzer-Services bestimmt:
- MS-Word
- MS-Excel und MS-Excel Makro-Programmierung
- MS-Multiplan
- MS-Chart

Informationsseminare

Diese Seminare sind für die Mitarbeiter des EDV-Benutzer-Service bestimmt:
- Einblick in Microsoft System-Software
- LAN-Manager und SQL-Server

Das Microsoft OS/2-Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Neben einem Überblick über den Intel 80286-Prozessor wird dann als Schwerpunkt dieses Seminars auf das Application Program Interface (API) eingegangen. In diesem Zusammenhang werden die Funktionen für Multitasking, Memory Management und Dynamic Linking ausführlich behandelt. Weitere Themen: Geräte- und Datei-Ein-/-Ausgabe. Die in diesem Seminar gezeigten Programmbeispiele sind in Microsoft C geschrieben.

Datum	Tag
Düsseldorf	
06./07.11.89	Mo./Di.
München	
02./03.10.89	Mo./Di.
04./05.12.89	Mo./Di.

Der Microsoft OS/2-Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das MS-OS/2-Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC den komfortablen Editor zu nutzen, Family-API-Programme zu schreiben und Device-I/O-Routinen zu erstellen sowie Multitasking-Funktionen zu nutzen und eigene Dynamic-Link-Bibliotheken zu erstellen; außerdem können sie die erweiterten Speicherverwaltungsmöglichkeiten des Intel 80286 nutzen und mit Hilfe des MS-OS/2 Memory Managers programmieren.

Datum	Tag
Düsseldorf	
08./09./10.11.89	Mi./Do./Fr.
München	
04./05./06.10.89	Mi./Do./Fr.
06./07./08.12.89	Mi./Do./Fr.

Das Microsoft Windows-Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Dieses Seminar behandelt die Microsoft Windows-Benutzerschnittstelle, die Microsoft Windows-Systemarchitektur, die Programmierwerkzeuge in der Praxis und Erstellung von Microsoft Windows-Programmen an Beispielen.

Datum	Tag
Düsseldorf	
23./24.10.89	Mo./Di.
München	
09./10.10.89	Mo./Di.
06./07.11.89	Mo./Di.
11./12.12.89	Mo./Di.

Der Microsoft Windows-Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das Microsoft Windows Einführungsseminar besucht haben.

In diesem Seminar erwirbt der Teilnehmer durch praktische Tätigkeit an einem PC die Fertigkeiten zur Programmierung unter Microsoft Windows. Dazu werden verschiedene Aufgaben gestellt und Übungen abgehalten. Als Hauptthemen dieses Seminars werden Übungen zu dem »Windowing«, der grafischen Programmierschnittstelle, den Benutzerschnittstel-

len und dem Dynamic Linking durchgeführt.

Datum	Tag
Düsseldorf	
25./26./27.10.89	Mi./Do./Fr.
München	
11./12./13.10.89	Mi./Do./Fr.
08./09./10.11.89	Mi./Do./Fr.
13./14./15.12.89	Mi./Do./Fr.

Microsoft Presentation Manager-Einführungs-Seminar
Dieses zweitägige Seminar für Windows- und C-Programmierer erläutert das Konzept des Presentation Managers und gibt einen Überblick über seine Programmierschnittstellen und Fähigkeiten.

Das Seminar behandelt die Benutzerschnittstelle des Presentation Managers, seine Architektur, die Programmierwerkzeuge in der Praxis und die Erstellung von PM-Programmen an Beispielen.

Datum	Tag
Düsseldorf	
16./17.10.89	Mo./Di.
18./19.12.89	Mo./Di.
München	
27./28.11.89	Do./Fr.

Microsoft Presentation Manager-Workshop
Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in C und eventuell in Windows besitzen sowie das Microsoft Presentation Manager Einführungsseminar besucht haben.

In diesem Seminar erwirbt der Teilnehmer durch praktische Tätigkeit an einem PC die Fertigkeiten zur Programmierung unter dem Microsoft Presentation Manager. Als Hauptthemen dieses Seminars werden das Application Program Interface (API) des Presentation Managers besprochen. Dazu werden verschiedene Aufgaben gestellt und Übungen abgehalten: Windowing, Dynamic Linking Libraries usw.

Datum	Tag
Düsseldorf	
18./19./20.10.89	Mi./Do./Fr.
20./21./22.12.89	Mi./Do./Fr.
München	
29./30.11/1.12.89	Mi./Do./Fr.

Microsoft LAN-Manager-Einführungs-Seminar
Dieses zweitägige Seminar für Netzanwender und Netzwerk-Softwareentwickler mit Programmiererfahrung in C vermittelt das neue zukunftsweisende Programmierkonzept, um den MS OS/2 LAN-Manager bedienen zu können.

In diesem Seminar werden die neuen Konzepte des LAN-Managers erörtert sowie neue und zukunftsweisende Programmiermethoden (hier ist die »distributed intelligence Method« zu nennen) vorgestellt. Weiterhin gibt das Seminar in seinem letzten Abschnitt eine Installations- und Bedienungsanleitung für den LAN-Manager und vermittelt so einen »roten Faden« für den Netzwerkanwender.

Datum	Tag
Düsseldorf	
02./03.10.89	Mo./Di.
23./24.11.89	Do./Fr
04./05.12.89	Mo./Di.
München	
13./14.11.89	Mo./Di.

Microsoft LAN-Manager-Workshop
Dieses dreitägige Seminar für PC-Softwareentwickler mit Programmiererfahrung in C, die am LAN-Manager Einführungsseminar teilgenommen haben, vermittelt die Fähigkeit zum schreiben von Programmen, die die grundlegenden Bestandteile des Microsoft LAN-Managers benutzen.

Der Schwerpunkt dieses Seminars liegt auf der Darstellung des API des LAN-Managers. Themengebiete wie »Named Pipes«, »Mailslots«, »Server-Client-Konzept« usw. werden mit Hilfe von Übungen und praktischen Programmierbeispielen erläutert.

Datum	Tag
Düsseldorf	
04./05./06.10.89	Mi./Do./Fr.
06./07./08.12.89	Mi./Do./Fr.
München	
15./16./17.11.89	Mi./Do./Fr.

Microsoft SQL-Server-Einführungs-Seminar
Dieses zweitägige Seminar für PC-Softwareentwickler erläutert die Struktur und das Konzept des Microsoft SQL-Server. Kenntnisse von MS OS/2 und dem MS LAN-Manager sollten vorhanden sein, SQL- und C-Kenntnisse sind von Vorteil.

Das Seminar gibt einen Überblick über die Struktur und den Aufbau des Microsoft SQL-Servers. Neben der Administration des Systems werden schwerpunktmäßig die Funktionsweise und die Einsatzgebiete des SQL-Servers behandelt.

Datum	Tag
Düsseldorf	
11./12.12.89	Mo./Di.
München	
23./24.10.89	Mo./Di.

Microsoft SQL-Server-Worksop
Dieses dreitägige Seminar für PC-Softwareentwickler mit Kenntnissen in C und SQL, die am Einführungsseminar und Workshop für OS/2 und den LAN-Manager teilgenommen haben, vermittelt die Fähigkeit zum Schreiben von Programmen, die die Funktionen des Microsoft SQL-Servers benutzen.

Der Schwerpunkt dieses Seminars liegt neben der Installation

und Administration des Systems auf der ausführlichen Erläuterung der DB-Library, der Programmierschnittstelle des Systems. Die Methoden zur Programmierung werden mit Hilfe von Übungen und praktischen Programmierbeispielen veranschaulicht.

Datum	Tag
Düsseldorf	
13./14./15.12.89	Mi./Do./Fr.
München	
25./26./27.10.89	Mi./Do./Fr.

Microsoft Excel-Makro-Programmierung

Dieses dreitägige Seminar wendet sich an Software-Entwickler, die mit Microsoft Excel schon einigermaßen vertraut sind.

Das Seminar gibt anfangs einen grundsätzlichen Überblick über Microsoft Excel, der für das Verständnis der Makroprogrammierung nötig ist. Durch einfachere Einzelbeispiele wird der Teilnehmer zunächst mit der Makroprogrammierung vertraut gemacht. Es folgt der Einstieg in komplexe Problemlösungen mit Makros, in denen u.a. sowohl selbstdefinierte Menüs und Dialogboxen als auch Funktions- und Befehlsmakros eingebunden sind. Weiterhin geht das Seminar auf die Microsoft Excel-Programmierumgebung ein, wobei auf die Möglichkeiten hingewiesen wird, Informationen aus anderer Software mit Makros zu verarbeiten (Datev-Datensätze) und wie mit Hilfe von Makrobefehlen die DDE-Schnittstelle von Microsoft Windows genutzt werden kann.

Datum	Tag
Düsseldorf	
09./10./11.10.89	Mo./Di./Mi.
15./16./17.11.89	Mi./Do./Fr.
27./28./29.11.89	Mo./Di./Mi.
München	
16./18./18.10.89	Mo./Di./Mi.

Leserumfrage:

Zwanzig QuickPascal zu gewinnen

Das neue Microsoft System Journal will stärker noch als bisher Werkzeug in der Hand professioneller Systementwickler, Programmierer und erfahrener Anwender sein. Deshalb wollen sich Herausgeber und Redaktion bemühen, den Anforderungen der Leser optimal gerecht zu werden. Mit den folgenden Fragen wollen wir mehr über unsere Leserschaft erfahren.

Welche Themen soll das Microsoft System Journal künftig verstärkt behandeln? Was interessiert unsere Leser besonders? Bei welchen Entscheidungen können wir Ihnen mit Rat zur Seite stehen? Müssen wir Rücksicht nehmen auf bei unseren Lesern eingesetzte Hardware? Die Ergebnisse dieser Umfrage sollen das Heft noch interessanter und lesenswerter machen. Wir versichern Ihnen Ihre Angaben vertraulich zu behandeln, nicht an Dritte weiterzuleiten und nur in anonymisierter Form zu bearbeiten. Über die Ergebnisse werden wir in einer unserer nächsten Ausgabe berichten.

Die ersten zehn Einsender erhalten als kleines Dankeschön und in Anerkennung ihrer schnellen Antwort einen Microsoft QuickPascal-Compiler. Damit aber auch Einsender eine Chance haben, die dieses Heft vielleicht erst nach einigen Tagen am Kiosk entdecken, werden wir weitere zehn QuickPascal-Compiler und 35 Bücher unter den restlichen Einsendern verlosen. Die Bücher wurden freundlicherweise von den verlagen Addison Wesley, Data Becker, Markt & Technik, Sybex, Systhema und Tewi zur Verfügung gestellt. Je früher Sie sich also entschließen diesen Fragebogen beantwortet an die Redaktion zurückzuschicken, desto größer sind Ihre Chancen mit uns gemeinsam abzuheben. Einsendeschluß ist der 31.10.1989. Der Rechtsweg ist ausgeschlossen. Bitte senden Sie den ausgefüllten Bogen an:

Synergy Verlag GmbH
Redaktion Microsoft System Journal
Theresienstr. 40
8000 München 2

1. Ich bin Abonnent des Microsoft System Journals:
   Ja O Nein O
   Wenn ja, seit wann? ____________
   Wenn nein, warum nicht? ____________

   Ich werde das System Journal künftig abonnieren. O
   Ich werde das System Journal regelmäßig am Kiosk kaufen. O
   Ich werde das System Journal gelegentlich am Kiosk kaufen. O
   Mir ist das System Journal zu teuer. O
   Mir ist das System Journal zu »populär«! O
   Mir ist das System Journal zu »schwierig«! O
   Sonstiges: ____________
   ____
   ____
   ____

2. Ich wurde auf das Microsoft System Journal aufmerksam durch
   eine Anzeige O
   Werbeversand O
   persönliche Empfehlung O

3. Ich interessiere mich für Artikel über

	Stark	Mittel	Wenig
MS-DOS	O	O	O
MS OS/2	O	O	O
Windows	O	O	O
LAN/SQL-Technologie	O	O	O
Macro Assembler	O	O	O
C	O	O	O
COBOL	O	O	O
FORTRAN	O	O	O
Pascal	O	O	O
BASIC	O	O	O
Hardware	O	O	O

4. Der C-Einsteigerkurs im Microsoft System Journal ist für mich
   interessant O
   uninteressant O
   zu schwierig O
   zu einfach O

5. Ich programmiere in
   BASIC O
   C O
   Pascal O
   COBOL O
   FORTRAN O
   Assembler O
   SQL O
   anderen Sprachen ____________

6. Ich beschäftige mich
   professionell O
   in meiner Freizeit O
   gar nicht O
   mit Programmierung

7. Ich arbeite als
   - Programmierer/Systementwickler O
   - Systemberater O
   - DV-Entscheider O
   - Berufstätiger in einem sonstigen DV-Beruf O
   - EDV-orientierte(r) Student(in) O
   - in einer nicht-EDV-orientierten Tätigkeit O

8. Ich bin
   - Selbständiger O
   - Leitender Angestellter O
   - Angestellter O
   - in Ausbildung O
   - Sonstiges ______________________

9. Ich arbeite
   - unter MS-DOS O
   - unter MS OS/2 O
   - UNIX/XENIX O
   - Macintosh-Betriebssystemen O
   - Sonstigen ______________________

10. Ich habe Erfahrung in der Entwicklung für
    - MS-DOS O
    - MS OS/2 O
    - Windows O
    - UNIX/XENIX O
    - Netzwerke O

11. Ich bin an Fortbildungsveranstaltungen des Microsoft System Journals zu folgenden Themen interessiert:
    - Entwicklung für MS OS/2 O
    - Entwicklung für Windows O
    - Entwicklung mit C O
    - Systemberatung O
    - Sonstiges ______________________

12. Ich bin an einem Meinungsaustausch mit anderen Entwicklern zu folgenden Themenkreisen interessiert:
    - Entwicklung für MS OS/2 O
    - Entwicklung für Windows O
    - Entwicklung mit C O
    - Systemberatung O
    - Sonstiges ______________________

13. Ich arbeite auf
    - einem 8086/88-PC O
    - einem 80286-PC O
    - einem 80386-PC O
    - einer Workstation O
    - einem Macintosh O

14. Ich beabsichtige in den kommenden zwölf Monaten die Anschaffung
    - neuer Programmierwerkzeuge O
    - neuer Standardapplikationen O
    - neuer Rechner O
    - sonstiger neuer Hardware (Karten, Drucker, etc...) O

15. Ich finde das Microsoft System Journal

interessant	O	uninteressant	O
aktuell	O	unaktuell	O
zu Software-lastig	O	zu Hardware-lastig	O
verständlich geschrieben	O	unverständlich	O
professionell	O	unprofessionell	O
übersichtlich	O	unstrukturiert	O
zu teuer	O	angemessen	O

16. Ich nutze die beigefügte Listing-Diskette:
    Ja O Nein O

17. Ich lese folgende andere Computer-Fachzeitschriften:
    - MS Journal/etc O
    - c't O
    - mc O
    - Design & Elektronik O
    - PC Woche O
    - Computerwoche O
    - Computerzeitung O
    - CHIP O
    - CHIP TOOL O
    - DOS International O
    - Computer Persönlich O
    - PC Magazin O
    - Zeitschriften aus den USA O
    - keine O
    - andere ______________________

18. Ich ziehe es vor, wenn die Dokumentation zu Microsoft Entwicklungswerkzeugen in folgender Sprache abgefaßt ist:
    - englisch O
    - deutsch O

Anschließend haben Sie hier Gelegenheit, Ihren Kommentar zum Microsoft System Journal aufzuschreiben - natürlich nur, wenn Sie wollen:

____

____

____

____

Wichtig! Bitte vergessen Sie nicht, Ihre Anschrift anzugeben – damit Ihr QuickPascal Sie auch erreicht, wenn Sie zu den Gewinnern gehören.

Datenschutzhinweis! Wir versichern Ihnen, daß wir Ihre Adresse und Ihre persönlichen Angaben nicht an Dritte weitergeben werden und daß die Auswertung Ihrer Antworten unabhängig von Ihrer Adressenangabe erfolgt.

Name: ______________________

Straße: ______________________

PLZ, Ort: ______________________

Land: ______________________

Wichtige Bücher über OS/2

Das wichtigste bei der Programmierung in einer neuen Betriebssystemumgebung sind gute Nachschlagewerke und Einführungen mit vielen instruktiven Beispielen. Für OS/2 sind solche Bücher nun erhältlich. Auf Englisch liegen vor: das dreibändige Nachschlagewerk *OS/2 Programmer's Reference* von Microsoft und die mit vielen Beispielen versehene Einführung *Programming the OS/2 Presentation Manager* von Charles Petzold. Beide Titel sind die Standardwerke in ihrem Bereich. Auf Deutsch gibt es zwei ähnlich geartete Werke, und zwar das *OS/2-Programmierhandbuch* von David E. Cortesi, das zum großen Teil als Nachschlagewerk konzipiert ist, und *Peter Norton's OS/2 für Insider* von Robert Lafore und Peter Norton.

Die Referenzhandbücher von Microsoft sind ein Muß für jeden OS/2-Programmierer, sie gehören deshalb auch zum Lieferumfang des Programmier-Toolkits für den OS/2 Presentation Manager (PTK). Sie sind jedoch auch separat über den Verlag Microsoft Press (in Deutschland über Vieweg) erhältlich. In der Kombination mit dem OS/2 Presentation Manager Softset von Microsoft gibt es damit einen preiswerten Einstieg in die Programmierung unter OS/2 für diejenigen, die bereits OS/2 und einen Microsoft-Compiler besitzen.

Im Band 1 werden die konzeptionellen Grundlagen der Programmierschnittstelle von OS/2 erläutert. Der Band 2 enthält eine umfassende alphabetische Beschreibung aller Programmierfunktionen des OS/2 Presentation Managers. Der Band 3 gleicht dem Band 2, enthält jedoch nur die Beschreibung der Basisfunktionen des OS/2-Kerns ohne den Presentation Manager. Wie gesagt, ohne diese Bände geht nichts.

Charles Petzold hat sich bereits mit seinem Buch *Programming Windows* und vielen Artikeln in Fachzeitschriften (zum Beispiel auch im *Microsoft System Journal*) einen sehr guten Namen gemacht. So wie man bei der Programmierung unter Windows nicht um sein Buch herumkommt, ist auch *Programming the OS/2 Presentation Manager* jedem Einsteiger, aber auch den fortgeschrittenen Programmierern, sehr ans Herz zu legen. Charles Petzold versteht es, komplizierte Themen verständlich und nachvollziehbar darzustellen. Dazu trägt auch bei, daß er Beispiele mit hohem Wiederverwendungswert einsetzt. Wer dieses Buch durchgearbeitet hat, kann sich als wirklichen Experten in der OS/2-Programmierung bezeichnen.

Doch rund zweieinhalbtausend Seiten englische Fachbegriffe sind nicht jedermanns Sache; viele Programmierer ziehen eine deutsche Dokumentation vor. Hier erscheinen nach und nach Übersetzungen umfangreicher amerikanischer Bücher. Markt & Technik hat kürzlich zwei gute Werke vorgelegt, *Peter Norton's OS/2 für Insider* von Robert Lafore und dem amerikanischen Software-Star Peter Norton und das *OS/2-Programmierhandbuch* von dem erfahrenen Fachautor David E. Cortesi. Beide Bücher behandeln leider noch nicht die Programmierung des Presentation Managers, sie beschränken sich noch auf die Basis-Funktionen von OS/2, wie sie auch schon in der Version 1.0 vorhanden waren. Dies ist bedauerlich, da OS/2 erst mit dem Presentation Manager seine ganze Stärke zeigt. Doch auch ohne Hinweise auf seine Programmierung sind die Bücher sehr nützlich, da sie viele der wesentlichen Konzepte von OS/2, wie Multitasking, Prozesse, Threads, Semaphore, Pipes, Queues, Monitore, Signale und dynamisches Linken, behandeln. Für Programmierer, die mit der meldungsgesteuerten Verarbeitung des Presentation Managers noch nicht zurechtkommen, ist dies vielleicht sogar der bessere Weg, sich mit diesen neuen Dingen vertraut zu machen.

Das Buch von Robert Lafore und Peter Norton ist als Tutorial konzipiert, das den Leser schrittweise zuerst an einfache Themen heranführt und dann erst nach und nach zu schwierigeren und komplexeren Themen. Es wendet sich also nicht an erfahrene Experten, sondern vor allem an solche Leser, die nur geringe Programmierkenntnisse haben. Besonderer Wert wird deshalb auf Beispiele gelegt. Alle Systemaufrufe werden anhand kleiner, lauffähiger Beispielprogramme erläutert. Die Autoren merken dazu ganz richtig an: »Die beste methode, etwas zu lernen, besteht darin, es zu tun.« Alle Beispielprogramme werden auf einer 1,2 Mbyte-Diskette mitgeliefert.

Etwas theoretischer geht David E. Cortesi an das Thema heran. Sein Buch gliedert sich in zwei Teile, die etwa jeweils die Hälfte ausmachen. Im ersten Teil werden die wichtigsten Eigenschaften von OS/2 beschrieben, der zweite Teil enthält eine umfangreiche Beschreibung aller Systemaufrufe des OS/2-Kernels. Hier wird für jeden Systemaufruf detailliert aufgeführt, welche Parameter erwartet werden, was der Aufruf macht, welche Fehlermöglichkeiten bestehen und wie der Aufruf und die verwendeten Datenstrukturen in C aussehen. Etwas unschön ist, daß sich bei der Übersetzung in die C-Beispiele zahlreiche Pascal-Kommentare (in geschweiften Klammern) eingeschlichen haben. Die Beschreibung eignet sich aber auch für Assemblerprogrammierer. Das Buch eignet sich gleichermaßen als Einführung in die Programmierung unter OS/2 als auch als Nachschlagewerk für die Systemaufrufe.

Einen Auszug aus dem *OS/2-Programmierhandbuch* von David E. Cortesi über die Mausprogrammierung können Sie ab der Seite 23 .

Microsoft: OS/2 Programmers Reference Including Presentation Manager, Volume 1. Redmond, Microsoft Press, 1989; 740 Seiten; ISBN 1-55615-220-5; $29.95.

Microsoft: OS/2 Programmers Reference Including Presentation Manager, Volume 2. Redmond, Microsoft Press, 1989; 557 Seiten; ISBN 1-55615-221-3; $29.95.

Microsoft: OS/2 Programmers Reference, Volume 3. Redmond, Microsoft Press, 1989; 430 Seiten; ISBN 1-55615-222-1; $19.95.

Charles Petzold: Programming the OS/2 Presentation Manager. Redmond, Microsoft Press, 1989; 850 Seiten; ISBN 1-55615-170-5; $29.95.

Robert Lafore, Peter Norton: Peter Norton's OS/2 für Insider. Ein Leitfaden zur Programmierung unter OS/2. Markt & Technik, 1989; 567 Seiten; ISBN 3-89090-646-X; DM 89,-. Mit 1,2 Mbyte Diskette.

David Cortesi: OS/2 Programmierhandbuch. Das unentbehrliche Nachschlagewerk für OS/2-Programmierer. Haar, Markt & Technik, 1989; 470 Seiten; ISBN 3-89090-732-6; DM 89,-.

VMM, Virtual Memory Manager:

Eine virtuelle Speicher-verwaltung in C

Durch die Größe des Hauptspeichers, wie er unter Microsoft OS/2 einem Programm zur Verfügung steht, werden Programmierer ziemlich verwöhnt. Es ist z.B. jetzt, nachdem Magma's Programmiereditor ME (nicht zu verwechseln mit dem Microsoft Editor) auf OS/2 portiert wurde, ohne weiteres möglich, Dateien zu bearbeiten, die größer sind als der vorhandene Hauptspeicher. Mit der DOS-Version des Editors war so etwas kaum zu bewerkstelligen. Um sehr große Dateien editieren zu können, mußte die DOS-Version von ME verbessert werden und als logische Konsequenz daraus entstand die virtuelle Speicherverwaltung VMM.

Ein weiterer Grund, der zur Entwicklung des VMM (engl. virtual memory manager) führte, sind die Unzulänglichkeiten der Funktion malloc aus der C-Laufzeit-Bibliothek. In verschiedenen Compiler-Versionen wurden die Funktionen zur Speicherbelegung unterschiedlich implementiert, alle Lösungen sind aber für den normalen Programmierer gleichermaßen undurchsichtig.

Aus der Sicht eines Programmierers ist es natürlich vorteilhaft, malloc dieses Undurchsichtige zu nehmen und die einzelnen Bestandteile von malloc in eigene Funktionen zur Speicherverwaltung einzubauen, die sich dann auch vernünftig mit dem Microsoft CodeView-Debugger austesten lassen. Das ist vor allem dann sinnvoll, wenn der begründete Verdacht besteht, daß ein Programm eine Reihe belegter Speicherbereiche verfälscht hat.

Nicht zuletzt sollte ME so erweitert werden, daß er nicht mehr durch einen kleinen Rest freien Hauptspeichers eines Systems eingeschränkt wird, denn es soll immer noch Leute geben, die mit 256-Kbyte-Maschinen arbeiten. Wenn man den Speicher berechnet, den allein DOS belegt, dazu die Größe von speicherresidenten (TSR-) Programmen und .EXE-Dateien addiert, kommt man schnell zu dem Ergebnis, daß auch bei 640 Kbyte Hauptspeicher kaum noch Platz für größere Anwendungen bleibt.

Ein Ziel bei der Entwicklung der virtuellen Speicherverwaltung für den ME-Editor war es, das System so einfach und offen zu gestalten, daß es ohne weiteres auch für andere Anwendungen verwendet werden kann. Außerdem sollte der VMM auch die Möglichkeiten von evtl. vorhandenem Expanded Memory nutzen können. (Obwohl der VMM für den ME-Editor EMS unterstützt, wird auf die Implementierung hier nicht weiter eingegangen. Die Problemlösung kann der Leser als eine Art Übung für sich ansehen.) Die API's von VMM sind in *Tabelle 1* dargestellt.

Der VMM muß im Large-Modell kompiliert werden, da wir Zeiger vom Typ far benutzen und einige Routinen der C-Bibliothek (z.B. memset) diesen Typ erwarten. Für ein anderes Speichermodell sind kleinere Anpassungen notwendig.

Im folgenden Text wird zur Erweiterung des VMM einiges angemerkt, das Sie aufnehmen und realisieren sollten. Wir würden uns freuen, wenn Sie solche Ideen durchführen und uns dann mitteilen, wie Sie bei der Umsetzung vorgegangen sind und wie sich diese Änderungen auf die Leistungsfähigkeit des VMM auswirken.

Initialisierung des VMM

Als erstes muß jedes Programm, das den VMM benutzt, die Initialisierungs-Funktion VMInit aufrufen. Damit wird eine Datei erzeugt, die zur Sekundärspeicher, d.h. Festplatte oder RAM-Disk,

`VMInit`
Initialisiert VMM und muß zum Beginn einer Anwendung, vor Belegung der ersten Speicherblöcke, aufgerufen werden.

`VMTerminate`
Beendet VMM und wird zum Abschluß einer Anwendung aufgerufen.

`char far *MemDeref(HANDLE h)`
Berechnet aus dem Wert der Handle h die Speicheradresse des zugehörigen Speicherblocks.

`HANDLE MyAlloc(unsigned size)`
Belegt im virtuellen Speicher einen Speicherblock der Größe size, füllt ihn mit Null und gibt eine Handle auf diesen Block zurück.

MyFree(HANDLE h)
Gibt den durch die Handle h referierten Block frei.

SetVMPageSize(int kbytes)
Verändert die Standardgröße von Seiten auf kbytes Kbyte. Z.B. setzt SetVMPageSize(16) die Defaultgröße auf 16 Kbyte.

MakePageDirty(HANDLE h)
Markiert die Handle, in der sich der durch Handle h referierte Speicherblock befindet, als verändert.

◂◂ *Tabelle 1: Die Programmierschnittstelle der virtuellen Speicherverwaltung VMM*

gebraucht wird, die sogenannte Swap-Datei. VMInit prüft nach, ob im Environment die Variable METEMP definiert wurde, um den vollständigen Pfadnamen für die Swap-Datei festzulegen. Ist METEMP undefiniert, wird die Swap-Datei im aktuellen Verzeichnis erzeugt. Die Benutzung von METEMP ist z.B. sinnvoll, wenn die Swap-Datei auf einer RAM-Disk liegen soll, um das Ein- und Auslagern (engl. swapping) von Speicherblöcken zu beschleunigen.

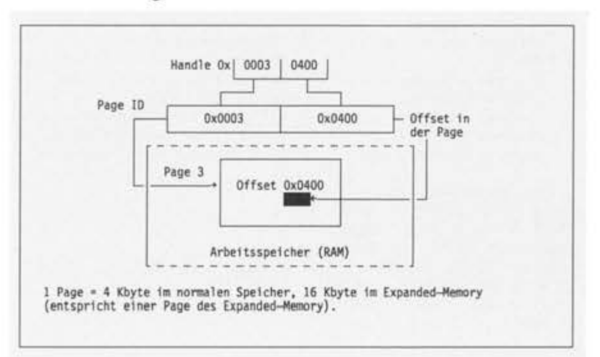
METEMP kann leicht durch Zufügen einer Zeile in die AUTOEXEC.BAT-Datei definiert werden:
`set METEMP =` <*Pfadname*>

Der Name der Swap-Datei wird mit Hilfe der Funktion mktemp, die in der C-Laufzeit-Bibliothek enthalten ist, festgelegt. Die Funktion benötigt als einzigen Parameter eine Zeichenkette als Muster und erzeugt damit eine eindeutige Dateibezeichnung. In diesem Fall wird das Muster VMXXXXXX an mktemp übergeben und die Funktion ersetzt alle X durch andere Zeichen, so daß der Dateiname sich von allen anderen im Zielverzeichnis unterscheidet. Zum Beispiel könnte mktemp an VMInit die Zeichenkette VM065291 zurückgeben und es würde eine Swap-Datei mit diesem Namen angelegt.

An dieser Stelle möchte ich eine der wesentlichen Einschränkungen des VMM erwähnen, nämlich die Begrenzung des Speichers zum Ein- und Auslagern auf die Größe des Swap-Mediums, d.h. RAM-Disk oder Festplatte (zusätzlich zum freien Speicher im Expanded Memory). Eine denkbare Verbesserung von VMM wäre also die Benutzung mehrerer Platten zum Ein- und Auslagern von Speicherblöcken, eventuell sogar über ein ganzes Netzwerk, also auf andere Rechner, z.B. einen File-Server. Dabei müssen aber unbedingt die DOS-Restriktionen, d.h. die maximale Anzahl gleichzeitig geöffneter Dateien einer Anwendung, beachtet werden!

Beenden des VMM

Bevor die Anwendung beendet werden kann, muß sie die Routine VMTerminate aufrufen, damit die Swap-Datei ordnungsgemäß geschlossen und gelöscht wird. Zusätzlich eingebaute Funktionen, z.B. statistische Analysen des Swapping oder Verteilung der Swap-Dateien über mehrere Platten oder Rechner, sollten an dieser Stelle ebenfalls abgeschloßen werden.



◂ *Bild 1: VMM-Handles*

Speicheranforderungen

Um Speicherplatz vom VMM anzufordern, wird die Funktion MyAlloc benutzt. Sie ersetzt zum einen die Standardroutine malloc, und da der belegte Speicherbereich mit '0' überschrieben wird, auch die Funktion calloc. An MyAlloc wird als Argument die Anzahl der benötigten Byte übergeben, zurück erhält man die Referenz auf eine Speicherseite, die ich im folgenden als Handle bezeichne.

Für diese Art von Referenz hat sich auch im Deutschen der Begriff Handle (im Englischen eigentlich soviel wie Henkel, Griff) weitgehend durchgesetzt, nicht zuletzt durch den häufigen Gebrauch in Microsoft Windows. Eine Handle ist einfach eine Bezeichnung für einen Speicherblock und wird von internen Routinen als Zeiger auf ein Objekt gebraucht. Der Wert dieser Variablen ist normalerweise für eine Anwendung mit dem VMM völlig bedeutungslos.

Der VMM verwendet einen vorzeichenlosen Long-Wert als Handle. Die oberen 16 Bit werden als Nummer des Speicherblocks, die unteren 16 Bit als Offset (mit 0 beginnend) innerhalb der

Seite interpretiert. Eine Handle mit dem Wert 00030400H zeigt also auf einen Speicherblock, der sich 400H Bytes vom Anfang des Blocks 3 befindet (*Bild 1*).

Mit dieser Konstruktion können also fast 64 Kbyte Seiten mit jeweils 64 Kbyte Inhalt gehandhabt werden, das sind insgesamt über 4 Gigabyte, die VMM adressieren kann. Selbst bei heutigen Plattenkapazitäten ein durchaus ausreichender Wert.

Aufbau einer Seite

Wie oben beschrieben, erwartet die Funktion MyAlloc als Argument die Größe des benötigten Speicherblocks, möglichst eine Potenz von 2, und legt, falls in den bereits existierenden Seiten nicht mehr genug Platz vorhanden ist, eine neue Seite an. Dafür benutzt er die Struktur PAGE aus VM.H (*Listing 1*).

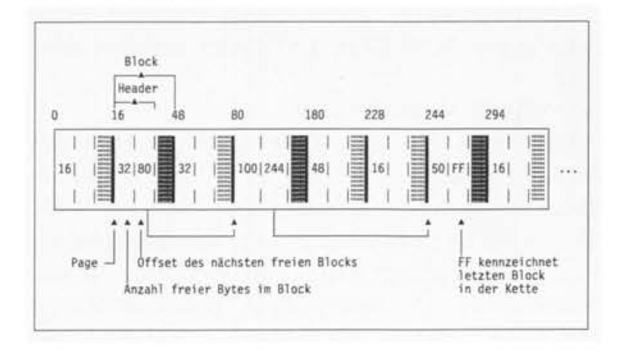
► *Listing 1: Struktur zur Definition eines Seitenkopfs*

```
typedef struct page
{
   struct page *next;          /* Zeiger auf nächste freie Seite  */
   char far *memaddr;          /* Speicheradresse der Seite       */
   unsigned long diskaddr;     /* Plattenadresse der Seite        */
   PAGEID   id;                /* Seitennummer                    */
   unsigned long LRUcount;     /* Zeit-Feld für LRU-Algorithmus   */
   unsigned pagesize;          /* Seitengröße in Byte             */
   unsigned freebyte;          /* Zgr auf 1. freien Speicherblock */
   unsigned bytesfree;         /* Anzahl freier Bytes der Seite   */
   unsigned maxcontigfree;     /* Bytes des größten freien Blocks */

   unsigned flags;
#define PAGE_IN_MEM     0x0001
#define PAGE_ON_DISK    0x0002
#define IS_DIRTY        0x0004
#define SET_PAGE_DIRTY(p)    ((p) -> flags != IS_DIRTY)
#define NON_SWAPPABLE   0x0008
#define PAGE_IN_EMM     0x0010
} PAGE;
```

Eines der ersten beiden Felder der Struktur PAGE, memaddr und diskaddr, adressiert die Handle, abhängig davon, ob sie sich im RAM oder im Sekundärspeicher (Festplatte oder RAM-Disk) befindet. Zur Adressierung im RAM ist ein far-Zeiger notwendig, für die Platte der Offset vom Anfang der Swap-Datei. Zusätzlich dazu enthält die Struktur noch die (eindeutige) Nummer der Seite, deren Größe (da der VMM mit verschiedenen Seitengrößen arbeiten kann), eine Bit-Maske, um den Status abzubilden (Steht die Seite im RAM und/oder auf der Platte? Wurde sie verändert?), einen Zeiger auf das erste freie Byte (zum Aufbau einer verketteten Liste freier Speicherblöcke) und ein Zeit-Feld des Swapping-Algorithmus für das Ein- und Auslagern. Hier wird der sogenannte LRU-Algorithmus benutzt, der die am längsten nicht verwendete (engl. least recently used) Seite bewegt. Außerdem beinhaltet die Struktur PAGE noch Felder für die Anzahl der freien Bytes der Seite insgesamt und für den größten zusammenhängenden freien Speicherblock. Diese beiden Werte ermöglichen das Verbinden benachbarter, freier Speicherblöcke. Auf diese Erweiterungsmöglichkeit für den VMM gehen wir später noch genauer ein.

Um die Leistungsfähigkeit der Anwendung zu erhöhen, kann man mit der Größe der Seiten experimentieren. Dazu dient die Routine SetVMSize, die allerdings direkt nach VMInit, zumindest aber vor der ersten Anlage von Seiten, und höchstens einmal im Programmablauf aufgerufen werden sollte, da der Swapping-Algorithmus eine konstante Seitengröße voraussetzt. Als Standardgröße für Seiten wurden 4 Kbyte für konventionellen Speicher und 16 Kbyte für Expanded Memory festgelegt. Damit kann der VMM das Speicherkonzept des Expanded Memory voll unterstützen. Innerhalb der Seiten werden freie Speicherblöcke durch eine verkettete Liste verwaltet, die durch die Struktur FREEINFO definiert ist: Der Kopf jedes freien (und angelegten) Speicherblocks enthält die Größe des Blocks und einen Zeiger auf den nächsten freien Block. Der letzte Block innerhalb der verketteten Liste hat den Zeigerwert FFFFH und ist damit als Listenende gekennzeichnet (eine typische Verkettung zeigt *Bild 2*). Da die Liste in keiner Weise geordnet oder strukturiert ist, sind natürlich Verbesserungen möglich, wie sie im Zusammenhang mit der Freigabe von Speicherblöcken genauer beschrieben sind. Beim Aufruf der Funktion MyAlloc wird die verkettete Liste durchlaufen, bis der erste Speicherblock mit der notwendigen Größe erreicht ist. Diese Suche übernimmt die Funktion FindNContigBytesFree. Befindet sich im Hauptspeicher kein passender Block, wird als nächstes auf der Festplatte oder der RAM-Disk weitergesucht. Wenn auch dort kein Speicherblock mit der benötigten Größe frei ist, wird die Funktion AllocPage aufgerufen, die für eine neue Seite Kopf und Pufferbereich anlegt. Danach hat MyAlloc den Zeiger auf eine Seite mit der genügenden Anzahl freier Bytes und es kann die verkettete Liste nach einem passenden freien Speicherblock durchsuchen, dessen Inhalt mit Null überschrieben wird. Zuletzt gibt MyAlloc an die aufrufende Routine die Handle des gefundenen Blocks zurück.



►► *Bild 2: Typischer Aufbau einer Speicher-Verkettung.*

Auslagern von Seiten

In dieser Version von VMM wird der Speicher für eine Handle durch den Aufruf der Routine _dos_allocmem (aus der Microsoft-C-Bibliothek) belegt, hinter der sich die DOS-Routine Int 21H mit Funktion 48H verbirgt. Durch die Benutzung

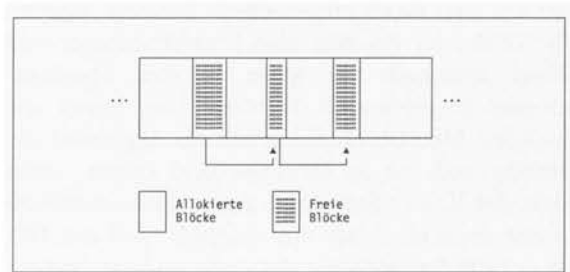
der DOS-Speicherroutinen kann auf die Funktion malloc der C-Laufzeit-Bibliothek völlig verzichtet werden. (Die Köpfe der Seiten werden zwar in dieser Version trotz allem mit malloc angelegt, aber das könnte genausogut mit DOS-Speicherroutinen erreicht werden.) Durch die Benutzung von _dos_allocmem tritt irgendwann die Situation ein, daß der Kopf für eine neue Seite existiert, aber für den Seitenrumpf kein Speicher mehr frei ist. Dann muß eine vorher angelegte Seite ausgelagert und abgespeichert werden, um Platz für den nächsten Speicherblock zu schaffen. Diesen Vorgang nennt man »swapping« oder »paging« auf die Swap-Datei. Die Belegung aller Sektoren der Swap-Datei ist in der Tabelle VMFile.slottable abgebildet. Ein Sektor ist entweder durch eine Seite belegt und die Nummer der Seite in die Tabelle eingetragen, oder der Sektor ist frei, was durch einen NULL-Eintrag gekennzeichnet wird. Soll also eine Seite das erste Mal auf die Swap-Datei ausgelagert werden, durchsucht der Swapping-Algorithmus die Belegungstabelle nach dem ersten NULL-Eintrag und schreibt die Handle in den entsprechenden Sektor. Die Auswahl des »richtigen« Algorithmus zur Bestimmung der Seite, die als nächste ausgelagert werden soll, ist dabei natürlich von außerordentlicher Bedeutung. Wird ein ungeeigneter Algorithmus implementiert, kann das dazu führen, das ein VMM extrem viel Zeit darauf verwendet, Seiten ein- und auszulagern. Dieses Phänomen wirkt sich stark auf die Effektivität der Speicherverwaltung aus und wird als »thrashing« bezeichnet. Es tritt z.B. dann auf, wenn Seiten ausgelagert werden, auf die sehr oft zugegriffen wird. Im VMM ist ein LRU-Algorithmus eingebaut, der die Seite auslagert, auf die die längste Zeit nicht zugegriffen wurde. Um feststellen zu können, welche Handle das ist, wird in der PAGE-Struktur das Zeit-Feld LRUcount mitgeführt, das bei jedem Zugriff auf die entsprechende Seite aktualisiert wird. Der LRU-Algorithmus prüft die Zeit-Felder aller Seiten und gibt einen Zeiger auf die Seite zurück, deren Kopf den ältesten Zeit-Eintrag enthält.

Eine mögliche Erweiterung des VMM ist die Verwendung einer doppelt verketteten Liste zur Verwaltung der freien Speicherblöcke und das Mitführen eines Zeigers auf das Ende dieser Liste. Wenn auf eine Seite zugegriffen und sie gleichzeitig an den Anfang der Liste gehängt wird, ist sichergestellt, daß die am längsten nicht verwendete Seite immer am Ende zu finden ist. Der LRU-Algorithmus muß dann nicht aufwendig die Zeit-Einträge aller Seiten überprüfen. Sinnvoll wäre auch die Kennzeichnung von Seiten als »nonswappable«, so daß sie nicht ausgelagert werden können. Das ist z.B. notwendig, wenn auch zum Anlegen von Köpfen der VMM benutzt wird, nicht die Funktion malloc. In diesem Fall muß aber auch das Auslagern von Seiten mit Köpfen und anderen, wichtigen Systeminformationen unterbunden werden, da sonst der VMM seine wichtigsten Werte auslagern und danach abstürzen würde. Der LRU-Algorithmus ist relativ einfach so zu verändern, daß er besonders gekennzeichnete Seiten ignoriert und somit nicht auslagert.

Belegungsregeln

Im allgemeinen unterscheidet man drei Regeln zur Einlagerung von Seiteen, je nachdem, ob der erste, nächste oder am besten passende Speicherblock belegt wird. Das hier verwendete Verfahren heißt »first fit«, da bei der Suche nach freiem Speichern der erste Block, der groß genug ist, eine neue Seite aufzunehmen, belegt wird. Dadurch wird die Suche in den meisten Fällen kurz.

Eine Verbesserung dieser Methode bezüglich der Anzahl der Suchschritte erreicht man, indem man sich die Position der letzten Einlagerung merkt und für den nächsten Block an dieser Stelle zu suchen beginnt. Darum heißt dieses Verfahren auch »next fit« oder »rotating first fit«, da Listenende und -anfang zusammengefaßt werden, also eine Art Ring entsteht.



◄ *Bild 3: Wenn freie Blöcke für die Allokierung zu klein sind, kommt es zur Speicherzerstückelung.*

Eine dritte, bekannte Einlagerungsregel durchsucht die gesamte Liste der freien Speicherblöcke, um den kleinsten Block zu finden, der gerade groß genug ist, den einzulagernden aufzunehmen. Diese Methode bezeichnet man als »best fit« und sie bietet den Vorteil, den Speicherbereich so gut auszunutzen, daß er nicht in kleinste Teile zerstückelt wird. Diese Zerstückelung ist ein großes Problem bei der Auslegung einer virtuellen Speicherverwaltung und entsteht dadurch, daß der Speicherblock mehr freie Bytes besitzt, als beim Einlagern benötigt werden. Dann wird nämlich nur ein Teil des Blocks belegt und der kleine Rest in die Liste der freien Speicherblöcke eingetragen. Dann ist es aber vorstellbar, daß dieser Rest zu klein ist, um überhaupt noch irgendwann belegt zu werden. Nehmen wir z.B. an, daß ein Block mit 24 Bytes benötigt und beim Durchsuchen der Liste nur ein freier Block mit 32 Bytes gefunden wird. Dann werden 24 Bytes von diesem Block belegt und die restlichen 8 Bytes gehen in die Liste der freien Blöcke (*Bild 3*). Wenn im weiteren Ablauf jeweils Blöcke mit mehr als 8 Byte benötigt werden, verbleibt der Restblock auf Dauer in der Liste. Das Problem besteht darin, daß der VMM bei jedem Suchlauf nach freien Blöcken diesen

Eintrag prüfen muß, und das führt normalerweise zu einer deutlich verschlechterten Laufzeit.

Das Problem kann man aber elegant umgehen, indem die Größe des angeforderten Speicherblocks auf die nächste Potenz von 2 aufgerundet wird. Die Anwendung erhält dann zwar einen größeren Speicherblock als sie braucht, aber die überzähligen Bytes erscheinen nicht in der Liste der freien Blöcke. Im Beispiel würde demnach statt eines 24 Byte großen Blocks einer mit 32 Byte belegt. Die restlichen 8 Byte wären übrig und die Liste der freien Speicherblöcke hätte sich nicht verändert.

Eine weitere Lösung des Problems, die wir weiter unten ebenfalls betrachten werden, ist das Umordnen von freien Speicherblöcken in größere Einheiten, das sogenannte »garbage collection«.

Umwandlung von Handles in Speicheradressen

Wie oben beschrieben, dient eine Handle für den VMM lediglich als Referenz auf einen Speicherblock, eine Anwendung kann damit nicht auf die Adresse des Blocks im Speicher zurückschließen. Um auf den Block zugreifen zu können, müssen die 32 Bit der Handle, also Handlenummer und Offset innerhalb der Seite, in eine Speicheradresse umgewandelt werden. Das leistet die Funktion MemDeref. Sie erhält als Argument die Handle, teilt sie in Nummer und Offset, sucht nach der ID der Seite und lagert diese eventuell in den Hauptspeicher ein. Danach wird der Offset auf die Basisadresse der Seite addiert und ein Zeiger auf den Block zurückgegeben.

Zusätzlich wird die Seite an den Anfang der verketteten Liste mit aktuellen Seiten gestellt, da der VMM davon ausgeht, daß auf sie mehrfach hintereinander zugegriffen wird. Das Umverketten einer erneut verwendeten Seite an den Anfang der Liste bringt dann natürlich Zeitvorteile bei nachfolgenden Zugriffen, da nicht lange in der Liste gesucht werden muß.

Soll z.B. die Zeichenkette XYZ in den Speicher kopiert werden, erreicht man das mit den folgenden Anweisungen:

```
#include "vm.h"
      .
      .
 HANDLE h;
 char far *s;  /* ACHTUNG: Large-Speichermodell notwendig */
      .
      .
 if ((h = MyAlloc(4)) == (HANDLE) NULL)
      .        /* nachfolgende Fehlerbehandlung */
      .
 if ((s = MemDeref(h)) != NULL)
    strcpy (s, "XYZ");
```

Veränderung von Seiten

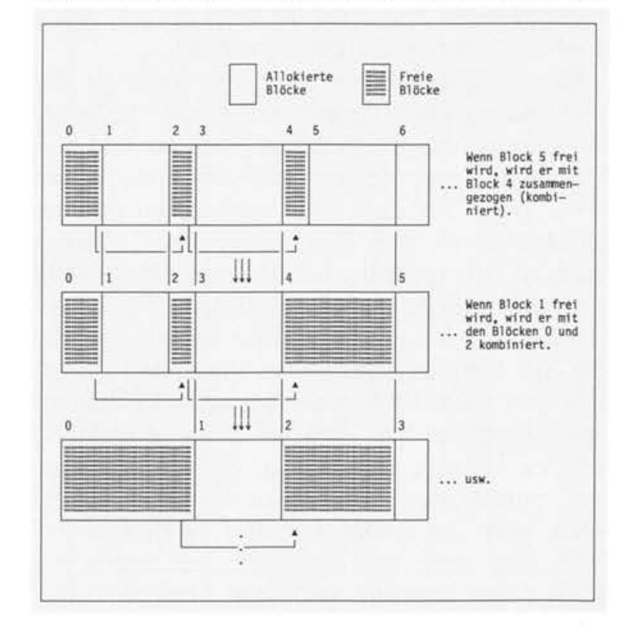
Wenn Sie sich im Listing die Funktion WritePage genau ansehen, stellen Sie fest, daß eine Seite nur dann auf die Platte zurückgeschrieben wird, wenn sie sich im Speicherraum befindet und sie seit dem Anlegen oder letzten Zurückschreiben verändert wurde. Wenn also der Seiteninhalt in Swap-Datei und Speicher identisch ist, kann er einfach überschrieben werden. Um die Veränderung von Seiten leicht zu erkennen, wird in der Struktur PAGE das Bit is_dirty mitgeführt. Es wird durch die Routine MakePageDirty gesetzt, die als Argument die Handle des Speicherblocks erhält und damit die zugehörige Seite identifiziert. Dadurch ist sichergestellt, daß eine veränderte Seite zum Auslagern ordentlich zurückgeschrieben wird. Soll z.B. die eben angelegte Zeichenkette XYZ, auf die die Handle h noch immer zeigt, in ABC umgewandelt werden, sind folgende Anweisungen notwendig:

```
s = MemDeref(h);
strcpy(s, "ABC");
MakePageDirty(h);
```

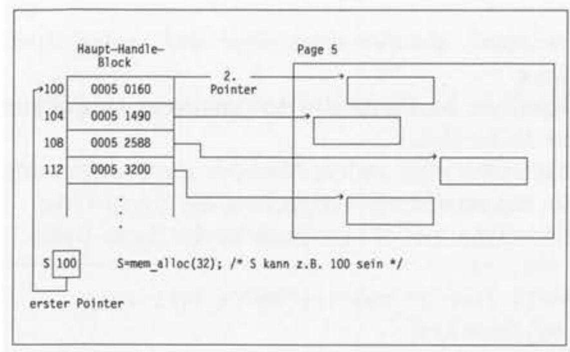
Freigabe von Speicherblöcken

Die Routine MyFree gibt nicht länger benötigte Speicherblöcke zur weiteren Benutzung wieder frei. Sie erhält als Argument die Handle, fügt den Block an die verkettete Liste freier Blöcke an und erhöht im Kopf FreeByte um die Anzahl der freigewordenen Byte. Zusätzlich werden benachbarte, freie Speicherblöcke miteinander verbunden und MaxContigFree auf die Anzahl Bytes des größten zusammenhängenden Blocks aktualisiert.

Um das Auffinden benachbarter freier Blöcke in der verketteten Liste zu beschleunigen, ist sie nach den Offsets der Blöcke sortiert. Dadurch muß in der Liste nur nach dem ersten Offset gesucht werden, der größer ist als der des neu freigegebenen Blocks. Dieser wird dann davor eingefügt (*Bild 4* zeigt ein typisches Zusammenfügen).



▸▸ *Bild 4: Zusammenziehung.*



◂◂ *Bild 5: Doppelte Verzeigerung.*

Dem aufmerksamen Leser ist natürlich schon aufgefallen, daß zwar Speicherblöcke von bis zu 64 Kbyte Größe angefordert werden (maximaler Argumentwert der Funktion MyAlloc), die angelegten Blöcke aber die Größe ihrer Seite nicht überschreiten können. Um diese Restriktion zu umgehen, kann der VMM so verändert werden, daß er die Seitengröße dynamisch, d.h. nach Anforderung sehr großer Speicherblöcke, erweitert.

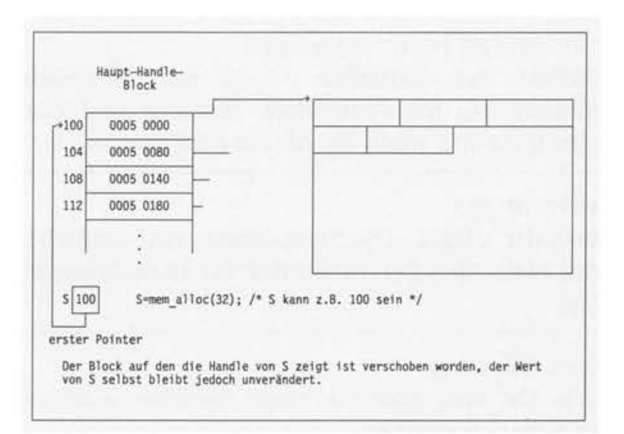
Verwendung indirekter Zeiger

Wie schon erwähnt, reduziert regelmäßiges Zusammenfügen freier Speicherblöcke die Zerstükkelung des Speicherraums. Vor dem Umordnen sind freie und belegte Blöcke vermischt und die verkettete Liste muß nach passenden, freien Plätzen durchsucht werden. Nach der Umordnung werden alle belegten Blöcke zusammengepackt und auf eine Seite der Handle kopiert, so daß ein zusammenhängender Block freien Speichers auf der anderen Seite der Handle entsteht. Wird danach Speicher für einen neuen Block angefordert, muß nur die Größe dieses freien Raumes geprüft werden.

Das größte Problem beim Umpacken entsteht dadurch, daß alle Zeiger aktualisiert werden müssen, wenn Blöcke im Speicher verschoben werden. Das ist für den VMM nicht möglich, da er dafür auf Zeiger der Anwendung zugreifen müßte. Um die Anpassung der Zeiger zu vermeiden, benutzen wir indirekte Zeiger. Diese Methode ist durch die Verwendung in Speicherverwaltungen sowohl im Macintosh, als auch in Microsoft Windows, bekannt geworden. Wenn also eine Anwendung einen Speicherblock benötigt, erhält sie nicht einen Zeiger direkt auf den Block, sondern einen Zeiger auf einen Zeiger auf den Block zurück (*Bild 5*).

Bild 6 zeigt die Vorteile von indirekten Zeigern beim Umpacken von Blöcken. Die indirekten Zeiger auf Speicherblöcke verändern sich nicht, obwohl die Blöcke verschoben werden. Da eine Anwendung nur diese Art von Zeigern erhält, die nicht angepaßt werden, muß der VMM die Anwendung bei einer Umpackaktion nicht benachrichtigen. Das Verschieben von Blöcken bleibt für die Anwendung unsichtbar. (Die Version 2.x von Microsoft Windows erlaubt einer Anwendung, sich beim Verschieben von Blöcken benachrichtigen zu lassen. Dazu muß lediglich das Flag GMEM_NOTIFY im Aufruf von GlobalAlloc gesetzt werden.)

Aus den Bildern 5 und 6 wird deutlich, daß für indirekte Zeiger eine zusätzliche Variable und ein zusätzlicher Speicherzugriff notwendig sind, um aus einer Handle eine Speicheradresse zu berechnen. Bei der Geschwindigkeit heutiger CPUs sollte das aber kein ernstzunehmendes Problem mehr darstellen, vor allem, wenn man die Zeitersparnis berücksichtigt, die sich ergibt, wenn das Durchsuchen der Liste wegfällt. Bedenkt man außerdem, daß der Speicherraum nicht mehr zerstückelt wird, ist klar, warum die Entwickler von Windows indirekte Zeiger benutzen.



◂ *Bild 6: Komprimierung mit doppelter Verzeigerung.*

Der McBride Allocator

Zum Schluß sehen wir und noch kurz eine andere virtuelle Speicherverwaltung an, VMEM von Blake McBride (*Tabelle 2*). Dieses Produkt wird mit vollständigem Quellcode ausgeliefert und benutzt ebenfalls indirekte Zeiger, um die Leistungsfähigkeit zu erhöhen. VMEM unterstützt zwar bisher nicht Expanded Memory, gleicht diesen Nachteil aber dadurch aus, daß die Swap-Datei umgepackt werden kann. Außerdem benutzt VMEM eine Erweiterung, die schon angesprochen wurde: die Verwaltung der virtuellen Speicherobjekte mittels einer doppelt verketteten Liste, so daß ein Zeiger immer auf das am längsten nicht verwendete Objekt weist.

Dem Benutzer stehen zwei Methoden zur Verfügung, die Swap-Datei umzuordnen. Einmal das Verschieben aller Objekte an den Anfang der Datei, so daß ein großes Loch am Ende der Datei entsteht. Dann die Verwendung zweier Dateien: Die angelegten Objekte werden aus der ersten in eine zweite Swap-Datei kopiert und die erste danach gelöscht. Obwohl beide Methoden die gleichen Ergebnisse liefern, bietet jede gegenüber der anderen einen bestimmten Vorteil. Beim Verschieben der Objekte innerhalb der Datei wird diese nicht komprimiert, bei der Benutzung zweier Dateien wird zum Kopieren kurzfristig der doppelte Platz auf der Platte belegt.

▸ *Tabelle 2: Routinenaufrufe der virtuellen Speicherverwaltung VMEM*

```
char far *VM_addr(VMPTR_TYPE voh,
int dirty, int fFreeze)
```
Berechnet aus Handle voh die zugehörige Speicheradresse. Wird der Speicherblock verändert, sollte die Variable dirty nicht gleich 0 sein.

fFreeze muß ungleich 0 sein, wenn der Block im Speicher nicht verschoben werden darf.

```
VMPTR_TYPE VM_alloc(long size, int fClear)
```
Belegt einen Speicherblock der Größe size. Wenn fClear nicht auf 0 gesetzt ist, wird der Block mit Null überschrieben.

```
VM_dcmps
```
Packt die Swap-Datei um. Die gewählte Methode wird durch VM_parm bestimmt.

```
int VM_dump(char *filename)
```
Sichert den aktuellen Stand des virtuellen Systems auf die Plattendatei filename und gibt eine 0 zurück, wenn die Aktion erfolgreich war.

```
void VM_end
```
Beendet VMEM. Die Swap-Datei wird gelöscht, der reale Speicher wird nicht für DOS freigegeben.

```
void VM_fcore
```
Wie VM_end, aber der reale Speicher wird an DOS zurückgegeben.

```
void VM_free(VMPTR_TYPE voh)
```
Gibt das durch Handle voh referierte Objekt frei.

```
int VM_rest(char *filename)
int VM_frest(char *filename)
```
Lädt den durch VM_dump gesicherten Stand des virtuellen Systems zurück. VM_rest und VM_frest unterscheiden sich im wesentlichen dadurch, daß VM_rest nur die benötigten Informationen zurücksichert und damit schneller als VM_frest arbeitet.

```
int VM_init
```
Initialisiert VMEM.

```
void VM_parm(long rmmax, long rmasize,
double rmcompf, long dmmfree, int dmmfblks,
int dmctype)
```
Setzt verschiedene Systemparameter von VMEM:
rmmax: maximale Größe des von VMEM benutzten, realen Speichers
rmasize: minimale Größe des von VMEM benutzten, realen Speichers
rmcompf: Kompressionsfaktor des realen Speichers
dmmfree: bestimmt das automatische Umpacken der Swap-Datei
dmmfblks: eine andere Methode zur Bestimmung des automatischen Umpackens der Swap-Datei
dmmctype: Art des Umpackens der Swap-Datei.

```
VMPTR_TYPE VM_realloc(VMPTR_TYPE voh,
long newsize)
```
Belegt einen durch die Handle voh referierten Speicherblock neu und gibt ein Handle darauf zurück.

```
long *VM_stat
```
Gibt verschiedene Werte des aktuellen Systemstatus zurück.

Zusätzlich dazu ermöglicht VMEM das Sichern und Zurückladen eines Abbilds des gesamten virtuellen Speichers, z.B. um den aktuellen Stand des Systems abzulegen, den Speicher an DOS freizugeben, ein weiteres Programm ablaufen zu lassen (etwa einen Compiler) und danach in der ersten Anwendung mit dem virtuellen Speicher weiterzuarbeiten, nachdem der gesicherte Stand wieder hergestellt wurde.

Beim genauen Hinsehen zeigt sich auch, daß das VMEM API etwas mehr Kontrolle über die Steuerparameter zuläßt, als es mit unserem VMM möglich ist. VMEM kann angegeben werden, ob er beim Anlegen neuer Blöcke Speicher löschen oder das Dirty-Bit beim Freigeben von Blöcken setzen soll. Außerdem können wesentliche Steuerparameter direkt verändert werden.

Die Verwendung einer virtuellen Speicherverwaltung als Ersatz für Speicher aus dem Heap ermöglicht einer Anwendung, flexibler mit großen Datenmengen zu operieren. Am Besten wäre es, virtuellen Speicher auf Betriebssystem-Ebene zu verwalten und Anwendungen auf höherem Niveau völlig davon abzutrennen, wie es z.B. Microsoft OS/2 verwirklicht. Da aber ein großer Teil des Marktes auch noch in naher Zukunft fest in der DOS-Welt verankert sein wird, sind virtuelle Speicherverwaltungen auf Anwendungsebene notwendig.

In Zukunft werden in allen Betriebssystemen virtuelle Speicherkonzepte implementiert sein, die dedizierte Chips zur Speicherverwaltung nutzen, wie sie schon jetzt von Intel und Motorola erhältlich sind. Hoffentlich dauert es bis dahin nicht mehr allzulang.

Marc Adler

◄◄ *Listing 2: VM.H*

```c
/****************************************************************/
/* Virtual Memory Manager           VM.H                        */
/*                                                              */
/* (C) Copyright 1988  Marc Adler/Magma Systems-All Rights Reserved*/
/*                                                              */
/* This software is for personal use only, and may not be used in */
/* any commercial product, nor may it be sold in any way.       */
/*                                                              */
/****************************************************************/

#define NO              0
#define YES             1

#define MAXSLOTS        1024
#define PAGESIZE        4096
#define EMM_PAGESIZE    16384
#define MAXPATHLEN      65

typedef unsigned PAGEID;
typedef unsigned long HANDLE;


/*
   The page header ....
*/
typedef struct page
{
  struct page *next;            /* chain to next free page in list */
  char far *memaddr;            /* memory address of the page block */
  unsigned long diskaddr;       /* disk address of the page block */
  PAGEID   id;                  /* page identifier */
  unsigned long LRUcount;       /* least-recently-used count    */
  unsigned pagesize;            /* how many bytes is this page   */
  unsigned freebyte;            /* index of 1st free byte in page */
  unsigned bytesfree;           /* # of bytes free in this page   */
  unsigned maxcontigfree;       /* max # of contiguous free bytes */

  unsigned flags;
#define PAGE_IN_MEM      0x0001
#define PAGE_ON_DISK     0x0002
#define IS_DIRTY         0x0004
#define SET_PAGE_DIRTY(p)      ((p)->flags |= IS_DIRTY)
#define NON_SWAPPABLE    0x0008
#define PAGE_IN_EMM      0x0010
} PAGE;


typedef struct disktable
{
#define SECTOR_FREE ((PAGE *) NULL)
  PAGE     *page;
} PAGE_DISK_ENTRY;


/*
   The free block header .....
*/
typedef struct freeinfo
{
  unsigned nextfree;  /* offset of next free entry (0xFFFF at end) */

#define FREELIST_END  0xFFFF

  unsigned bytesfree; /* # of bytes in this chunk */
} FREEINFO;


/*
  This contains info about the swap file...
*/
typedef struct vmfile
{
  char filename[MAXPATHLEN];     /* name of the VM file */
  int  fd;                       /* file handle to the VM file */
  PAGE_DISK_ENTRY slottable[MAXSLOTS];
} VMFILE;

/* External declarations */
extern char      VMInitialized;
extern unsigned TotalPages;
extern PAGE      *PageList;
extern VMFILE    VMFile;
extern unsigned VMPageSize;

/*global*/  int VMInit(void);
/*global*/  int VMTerminate(void);
/*global*/  char *AllocPageText(struct page *, unsigned int );
/*global*/  struct page *AllocPage(void);
/*global*/  int ReadPage(struct page *);
/*global*/  int WritePage(struct page *);
/*global*/  int SwapoutPage(struct page *,int );
/*global*/  int SwapinPage(struct page *);
/*global*/  struct page *FindLRUPage(void);
/*global*/  int FindSlotFree(void);
/*global*/  HANDLE MyAlloc(unsigned int );
/*global*/  void MyFree(HANDLE);
/*global*/  struct page *FindNBytesFree(unsigned int );
/*global*/  struct page *FindNContigBytesFree(unsigned int );
/*global*/  int CompactifyPage(struct page *);
/*global*/  char far *MemDeref(HANDLE);
/*global*/  PAGE *PageDeref(HANDLE);
/*global*/  char *mymalloc(unsigned);
```

◄ *Listing 3: VM.C*

```c
/****************************************************************/
/* Virtual Memory Manager           VM.C                        */
/*                                                              */
/* (C) Copyright 1988 Marc Adler/Magma Systems-All Rights Reserved */
/*                                                              */
/* This software is for personal use only, and may not be used in  */
/* any commercial product, nor may it sold in any way.          */
/*                                                              */
/****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <malloc.h>
#include <fcntl.h>
#include <io.h>
#include <sys\types.h>
#include <sys\stat.h>
#include "vm.h"


/* macro to return a ptr to a freeinfo structure */
#define FREEPTR(pg, offset)   (FREEINFO *) (pg->memaddr + offset)

unsigned TotalPages = 0;       /* Total # of pages allocated */
char VMInitialized = NO;       /* TRUE if the VMM has be initialized */
PAGE *PageList = NULL;         /* Linked list of pages */
VMFILE VMFile;                 /* VM File info */
unsigned VMPageSize = PAGESIZE; /* Pagesize of a VM block */
unsigned long LRUclock = 0;     /* Clock used for LRU swapping */
unsigned Emergency = NO;        /* TRUE if DOS has no more memory */



/* VMInit
     Initialize the Virtual Memory Manager (VMM).
*/
VMInit()
{
  char *mktemp();
  char *pascal strend();
  int  i;
  char *tempdir;

  /*
     Create the VM swap file. Try to use the path specified in the
     METEMP environment variable. If METEMP was not specified, use
     the current directory.
  */
  VMFile.filename[0] = '\0';
  if ((tempdir = getenv("METEMP")) != NULL)
  {
    char *end = strend(strcpy(VMFile.filename, tempdir));
    if (end[-1] != '\\')                   /* add the backslash */
    {
      *end = '\\';
      *++end = '\0';
    }
  }

  strcat(VMFile.filename, mktemp("VMXXXXXX"));
  if ((VMFile.fd = open(VMFile.filename,
                    O_RDWR|O_TRUNC|O_CREAT|O_BINARY,
                    S_IWRITE|S_IREAD)) < 0)
  {
    err("VMM - Cannot initialize VM system");.
    die(-1);
  }

  for (i = 0;  i < MAXSLOTS;  i++)
    VMFile.slottable[i].page = NULL;

  VMInitialized = YES;
  return 0;
}


/* VMTerminate
     Close and delete the VM swap file.
*/
VMTerminate()
{
  if (VMInitialized)
  {
    close(VMFile.fd);
    unlink(VMFile.filename);
    VMInitialized = NO;
  }
}


/* MemDeref
     main routine for dereferencing a VM handle.
*/
char far *MemDeref(h)
  HANDLE h;
{
  int  pid, offset;
  PAGE *p, *prevp;

  /* Separate the handle into the page id
     and the offset within that page */
```

Listing 3: Fortsetzung

```
  pid = (int) ((h >> 16) & 0x0000FFFF);
  offset = (int) (h & 0x0000FFFF);

  /* Search the linked list of pages for
     the page with id 'pid'. */

  for (prevp=p=PageList; p && p->id != pid; prevp=p, p=p->next)
    ;

  if (!p)
    return (HANDLE) NULL;

  /* If the page is not in conventional memory, swap it in. */
  if (!(p->flags & PAGE_IN_MEM))
    SwapinPage(p);

  /* Update the LRU count and bring the
     page to the front of the pagelist */

  p->LRUcount = LRUclock++;
  if (p != PageList)
  {
    prevp->next = p->next;
    p->next = PageList;
    PageList = p;
  }

  /* Return the absolute address which the handle references. */
  return p->memaddr + offset;
}


MakePageDirty(h)
  HANDLE h;
{
  PAGE *p;
  if ((p = PageDeref(h)) != NULL)
    SET_PAGE_DIRTY(p);
}


PAGE *PageDeref(h)
  HANDLE h;
{
  int  pid, offset;
  PAGE *p;

  /* Separate the handle into the page id
     and the offset within that page */

  pid = (int) ((h >> 16) & 0x0000FFFF);
  offset = (int) (h & 0x0000FFFF);

  /* Search the linked list of pages for the page with id 'pid'. */
  for (p = PageList;  p && p->id != pid;  p = p->next)
    ;
  return p;
}


/* AllocPage
     Allocates a new page and links it to the page list.
*/
PAGE *AllocPage()
{
  PAGE *p;
  char *s;
  FREEINFO *f;

  /* Make sure that the VMM is ready. */
  if (!VMInitialized)
    VMInit();

  /* Allocate the page structure from the heap */
  if ((p = (PAGE *) calloc(sizeof(PAGE), 1)) == NULL)
    return NULL;

  /* Allocate the page's text buffer from DOS or EMM */
  if ((s = AllocPageText(p, VMPageSize)) == NULL)
    return NULL;

  /* Fill the page structure. */
  p->memaddr        = s;
  p->flags         |= PAGE_IN_MEM;
  p->id             = ++TotalPages;
  p->LRUcount       = LRUclock++;
  p->pagesize       = VMPageSize;
  p->freebyte       = 0;
  p->bytesfree      = VMPageSize;
  p->maxcontigfree  = VMPageSize;


  /*
     Set up the page's initial free node.

            p                    memaddr
      _______________       _______________
      | freebyte | 0 |----> |nextfree |-1 |
      _______________       _______________
      | bytesfree|4K |      |bytesfree|4K |
      _______________       _______________
  */
  f = (FREEINFO *) s;
  f->bytesfree = VMPageSize;
  f->nextfree  = FREELIST_END;

  /* Link the page to the head of the pagelist. */
  p->next = PageList;
  PageList = p;

  return p;
}


/* AllocPageText
 * Allocs a block of 'size' bytes to be used as the page's buffer.
 */

char far *AllocPageText(page, size)
  PAGE    *page;
  unsigned size;
{
  PAGE *p;
  char far *s;

  /* Use DOSALLOC() to allocate the 4K page buffer */
  while ((s = mymalloc(size)) == NULL /* || test_swap */)
  {
    /* Swap out the Least Recently Used page.
       Return the LRU page ptr. */

    if ((p = FindLRUPage()) == NULL)
    {
      err("VMM - FindLRUPage() can't find a page to swap");
      return NULL;
    }

    /* Use the swapped-out page's text buffer as the new buffer. */
    s = p->memaddr;
    SwapoutPage(p, NO);
    memset(s, 0, size);
    break;
  }

  return s;
}



/*
  ReadPage()
  This is called from MemDeref() when the referenced page is not in
  conventional memory.

  If the page is already in memory, then just return;

  Seek to the disk address and read the proper number of bytes
  mark the page as being in memory

*/

ReadPage(p)
  PAGE *p;
{
  char *s;

  if (p->flags & PAGE_ON_DISK)
  {
    /* Obtain a text buffer which the
       swapped page will be read into. */

    if ((s = AllocPageText(p, p->pagesize)) == NULL)
      return -1;

    /* Seek to the proper place and read the page. */
    lseek(VMFile.fd, (long) (p->pagesize * p->diskaddr), 0);
    if (read(VMFile.fd, p->memaddr = s, p->pagesize) != p->pagesize)
    {
      err("FATAL ERROR! read() failed in ReadPage()");
      die(-1);
    }

    /* Say that the page is now in conventional
       memory as well as on disk. */

    p->flags |= PAGE_IN_MEM;
    p->LRUcount = LRUclock++;
  }

  return 0;
}


/*
  WritePage()

  If the page is on disk or it's in memory and there
  is a copy out on disk and the page isn't dirty return

  Find a free sector in the VM file
  Seek to the free sector and write the page
  Clear the dirty bit
  make an entry in the page/disk table and put the page in there
*/
```

```
WritePage(p)
  PAGE *p;
{
  unsigned sector;

  /* If the page is not in memory, then don't write it to disk */
  if (!(p->flags & PAGE_IN_MEM))
    return 0;

  /* The page has a disk sector allocated to it already */
  if ((p->flags & PAGE_ON_DISK) && !(p->flags & IS_DIRTY))
    return 0;  /* the in-mem copy is the same as the copy on disk */

  if ((sector = FindSlotFree()) == 0xFFFF)
  {
    err("VMM - No more slots free.");
    return -1;
  }

  /* See to the sector and dump the page text. */
  lseek(VMFile.fd, (long) (sector * (long) p->pagesize), 0);
  if (write(VMFile.fd, p->memaddr, p->pagesize) != p->pagesize)
  {
    err("FATAL ERROR! write() failed in WritePage()");
    exit(-1);
  }

  VMFile.slottable[sector].page = p;
  p->diskaddr = (long) sector;
  p->flags |= PAGE_ON_DISK;
  p->flags &= ~IS_DIRTY;
  p->LRUcount = LRUclock++;
}


SwapOutAllPages()
{
  PAGE *p;
  for (p = PageList;  p;  p = p->next)
    SwapoutPage(p, YES);
}



/*
  This is called from AllocPageText() when there
  is no more DOS memory free and no more Expanded
  memory free to allocate a page.
*/

SwapoutPage(p, free_it)
  PAGE *p;
{
  WritePage(p);
  if (free_it)
  {
    if (p->flags & PAGE_IN_MEM)
      my_free(p->memaddr);
  }
  p->flags &= ~PAGE_IN_MEM;
}



SwapInAllPages()
{
  PAGE *p;
  for (p = PageList;  p;  p = p->next)
    SwapinPage(p);
}

SwapinPage(p)
  PAGE *p;
{
  ReadPage(p);
}


/*
   FindLRUPage
   Finds the Least Recently Used page and returns a pointer to it.
*/

PAGE *FindLRUPage()
{
  PAGE *p;
  PAGE *retp = NULL;
  unsigned long minLRU = 0xFFFFFFFF;

  for (p = PageList;  p;  p = p->next)
  {
    if ((p->flags & PAGE_IN_MEM) &&
         p->LRUcount <= minLRU)        /* && IS_SWAPPABLE */
    {
      retp = p;
      minLRU = p->LRUcount;
    }
  }

  return retp;
}
```

Listing 3: Fortsetzung

```
/*
  FindSlotFree
     returns the first unused entry in VMFile.slottable
*/
FindSlotFree()
{
  register int i;

  for (i = 0;  i < MAXSLOTS;  i++)
    if (VMFile.slottable[i].page == NULL)
      return i;
  return -1;
}


/**********************************************************/
/*                                                        */
/*        Routines to allocate mem from a page            */
/*                                                        */
/**********************************************************/

/*
  MyAlloc(n)

  We want to allocate n bytes from the system.

  Make sure that n is less than VMPageSize.
  Make sure that n is a mutiple of 8.

  Find a page which has a contiguous block of n
  bytes free. Give preference to a page which is
  already in memory.

  If a page has n bytes free but not contiguous,
  then we will do compaction on the page.

  If there is still no page with n bytes free,
  then allocate a new page.

  At this point, we have a page 'p' with at least
  n bytes free.

  Decrease the bytes-free count of the page.
  Link the unallocated bytes onto the page's free list.
  Update the MaxContiguousFree count of the page.

  Return the Page/Offset value to the caller.
*/

HANDLE MyAlloc(needed)
  unsigned needed;
{
  FREEINFO *f, *prevf, *newf;
  PAGE     *p, *FindNBytesFree();
  unsigned offset, bytes_needed, bytes_left;
  unsigned maxcontig, new_offset;
  HANDLE h;

  needed = (needed + sizeof(FREEINFO));
  if (needed <= 0 || needed > VMPageSize)
  {
    return (HANDLE) NULL;
  }

  /* Return a page that has at least 'needed' bytes free */
  if ((p = FindNContigBytesFree(needed)) == NULL)
    return (HANDLE) NULL;

  /* Traverse the list of free chunks in
     the page and find the first  */


  /* chunk with 'needed' bytes free. 'F'
     will be the addr of the chunk. */

  maxcontig = 0;
  offset = p->freebyte;

  for (f = (FREEINFO *) (p->memaddr + p->freebyte);
           f->bytesfree < needed;
           f = (FREEINFO *) (p->memaddr + offset))
  {
    prevf = f;   /* save ptr to previous chunk for linking */
    if ((offset = f->nextfree) == FREELIST_END)

      /* If we get here, then there is not 'needed'
         contiguous bytes free */

      return (HANDLE) NULL;
    maxcontig = max(maxcontig, f->bytesfree);
  }

  /* Unlink and relink f (make sure there
     is enough room for the link) */

  bytes_left = f->bytesfree - needed;
  if (bytes_left <= sizeof(FREEINFO))  /* less than 4
                                          bytes remaining? */
  {
    needed = f->bytesfree;  /* no room for the link - */
    bytes_left = 0;         /* alloc the whole thing  */
  }
```

VMM in C

Listing 3: Fortsetzung

```
  /*
     If the chunk that we are allocating is smaller than the
     page's biggest free chunk, then there should be no change
     to the max. We set maxcontig equal to the page's maxcontig
     so that we won't traverse the rest of the free chain below.
   */

  if (f->bytesfree < p->maxcontigfree)
    maxcontig = p->maxcontigfree;

  if (bytes_left == 0)
  { /* We allocate the entire chunk */
    if (offset == p->freebyte)   /* The chunk is the 1st free one */
      p->freebyte = f->nextfree; /* so, relink the head pointer.  */
    else
      prevf->nextfree = f->nextfree;     /* Relink the previous */
    new_offset = f->nextfree;            /* chunk to next       */
  }

  else /* if (bytes_left) */
  { /* There is some space left over in the chunk */
    if (offset == p->freebyte)
      p->freebyte = offset + needed;
    else
      prevf->nextfree = offset + needed;
    /* Create a new chunk from the remaining bytes */
    newf = (FREEINFO *) (p->memaddr + offset + needed);
    newf->nextfree = f->nextfree;
    newf->bytesfree = bytes_left;
    maxcontig = max(maxcontig, newf->bytesfree);
    new_offset = offset + needed;
  }

  /* Now we traverse the rest of the freelist looking for a chunk */
  /* with more bytes free than maxcontig.                         */
  if (maxcontig < p->maxcontigfree && new_offset != FREELIST_END)
  {
    FREEINFO *f2;
    for (f2 = (FREEINFO *) (p->memaddr + new_offset);
         f2->bytesfree < p->maxcontigfree;
         f2 = (FREEINFO *) (p->memaddr + new_offset))
    if ((new_offset = f2->nextfree) == FREELIST_END)
      break;
    maxcontig = max(maxcontig, f2->bytesfree);
  }

  p->bytesfree -= needed; /* decrease # of free bytes in the page */
  p->maxcontigfree = maxcontig;
  f->bytesfree = needed;  /* set the length field
                             in the returned chunk */

  /* Clear the allocated memory to zeroes */
  memset(p->memaddr + offset + sizeof(FREEINFO),
         '\0', needed-sizeof(FREEINFO));

  /* Return offset */
  h = (HANDLE) (((long) p->id) << 16) |
      (long) (offset + sizeof(FREEINFO));
  return h;
}


/* Return a pointer to a page with N bytes free */
PAGE *FindNContigBytesFree(needed)
  unsigned needed;      /* # of bytes needed */
{
  PAGE *diskpage = NULL;
  PAGE *p;

  /*
     Traverse the page list looking for a page with the needed
     amount of bytes free in one contiguous chunk.
  */
  for (p = PageList;  p;  p = p->next)
  {
    if (p->maxcontigfree >= needed)
    {
      if (p->flags & PAGE_IN_MEM)
      { /* If the page is in memory, then return it immediately */
        return p;
      }
      else if (p->flags & PAGE_ON_DISK)
      { /* If the page is on disk, then mark it as the candidate */
        if (!diskpage) diskpage = p;
      }
    }
  }


  /*
     At this point, we have no memory-based page with
     n contiguous bytes, and possibly a disk-based page
     with n contiguous bytes.
  */

  if (diskpage)
  {
    SwapinPage(diskpage);
    CompactifyPage(diskpage);
    return diskpage;
  }
  else
  {
    /* No disk page had the needed bytes!!!
       Try to allocate a new page! */

    return AllocPage();
  }
}


CompactifyPage(p)
  PAGE *p;
{
  /*
     This routine has nothing in it yet.... Maybe one day,
     we'll add compaction....
   */
}



void MyFree(h)
  HANDLE h;
{
  char *s;
  int  pid;
  PAGE *pg;
  FREEINFO *f, *hFree, *prevf;
  unsigned fOffset, hOffset, prevfOffset;

  /* Swap the page in if necessary */
  if ((s = MemDeref(h)) == NULL)
    return;

  pid = (int) ((h >> 16) & 0x0000FFFF);
  hOffset = (int) (h & 0x0000FFFF);
  hOffset -= sizeof(FREEINFO);
  for (pg = PageList;  pg && pg->id != pid;  pg = pg->next)  ;
  SET_PAGE_DIRTY(pg);


  hFree = FREEPTR(pg, hOffset);

  pg->bytesfree += hFree->bytesfree;

  /* See if we have an empty free list */
  if (pg->freebyte >= pg->pagesize)
  {
    /*
         pg          h
      freebyte——>nextfree——>X
    */
    pg->freebyte = hOffset;
    pg->maxcontigfree = hFree->bytesfree;
    hFree->nextfree = FREELIST_END;
    return;
  }

  /* Traverse free chain until we get a free chunk whose address */
  /*  is larger than the address of the chunk about to be freed. */
  for (f = FREEPTR(pg, (fOffset = pg->freebyte));
       hOffset > fOffset && f->nextfree != FREELIST_END;
       prevfOffset=fOffset, prevf=f,
       f = FREEPTR(pg, (fOffset = f->nextfree)))
    ;

  /* See if we should insert the new chunk after the tail */
  /*
      100         200
       f           h
    nextfree——>nextfree——>X

  */
  if (hOffset > fOffset) /* we got here cause we hit the end */
  {
    /* Try to combine chunks f and h */
    if (fOffset + f->bytesfree == hOffset)
    {
      f->bytesfree += hFree->bytesfree;
      /* We should examine the block after f for coalescing */
    }
    else        /* Can't combine */
    {
      hFree->nextfree = FREELIST_END;
      f->nextfree = hOffset;
      /* We should examine the block after h for coalescing */
set_f_to_h:
      f = hFree;
      fOffset = hOffset;
      pg->maxcontigfree = max(pg->maxcontigfree, hFree->bytesfree);
    }
  }


  /* See if we should insert before the first free chunk */
  else if (pg->freebyte == fOffset)
  {
    /*
        pg->freebyte = 300

        200    300
         h——>f
    */
```

Listing 3: Ende

```
    hFree->nextfree = fOffset;
    pg->freebyte = hOffset;
    /* We should examine the block after h for
       coalescing */
    goto set_f_to_h;
  }


  /* We must be inserting between two existing
     free chunks */
  else
  {
    /*
       100          200         300
      prevf          h           f
     nextfree——>nextfree——>nextfree——>
    */
    /* See if we can coalesce prevf and h */
    if (prevfOffset + prevf->bytesfree ==
        hOffset)
    {
      prevf->bytesfree += hFree->bytesfree;
      /* We should examine the block after prevf
         for coalescing */
      f = prevf;
      fOffset = prevfOffset;
      pg->maxcontigfree = max(pg->maxcontigfree,
                              prevf->bytesfree);
    }
    else
    {
      hFree->nextfree = prevf->nextfree;
      prevf->nextfree = hOffset;
      /* We should examine the block after
         h for coalescing */
      goto set_f_to_h;
    }
  }

  /* We want to examine the block after
     chunk f to see if we can coalesce */

  if (f->nextfree != FREELIST_END  &&
        fOffset + f->bytesfree == f->nextfree)
  {
    hFree = FREEPTR(pg, f->nextfree);
    f->nextfree = hFree->nextfree;
    f->bytesfree += hFree->bytesfree;
    pg->maxcontigfree = max(pg->maxcontigfree,
                            f->bytesfree);
  }
}


char far *mymalloc(size)
  unsigned int size;
{
  char *s;
  unsigned seg;
  unsigned paras;

  /* We know that we have no DOS memory left */
  if (Emergency)
    return NULL;

  /* Convert bytes to paragraphs */
  paras = (size + 15) >> 4;

  /* Let DOS to do the allocation */
  if (_dos_allocmem(paras, &seg) != 0)
  {
    Emergency++;
    return NULL;
  }

  /*
    Convert the alocated memory into a far
    address and clear it out
  */

  FP_SEG(s) = seg;
  FP_OFF(s) = 0;
  memset(s, 0, paras << 4);
  return s;
}


my_free(s)
  char far *s;
{
  /* The memory must be returned to DOS */
  _dos_freemem(FP_SEG(s));
  Emergency = FALSE;
   /* we can alloc some more DOS memory */
}


SetVMPageSize(kbytes)
  int kbytes;
{
  VMPageSize = (unsigned) kbytes * 1024;
}
```

Eine komfortable Benutzeroberfläche in C (Teil 5):

Erstellung und Verwaltung von SAA-Dialogboxen

Dialogboxen – dieser Begriff steht als Synonym für die Art und Weise, wie SAA-Applikationen Daten vom Anwender über Tastatur und Maus empfangen. Und da auch PCs im wesentlichen nichts anderes als Datenverarbeitungsanlagen sind, kommt den Dialogboxen als Teil des SAA-Konzepts naturgemäß eine große Bedeutung zu. Grund genug, auch diesen Bestandteil des weit gefächerten SAA-Konzepts im Rahmen unserer SAA-Serie genauer unter die Lupe zu nehmen und eine mögliche Umsetzung in C vorzuführen.

Wenn Sie diese Serie und damit unseren Streifzug durch die SAA-Welt schon ein wenig länger verfolgen und sich an die bisherigen Folgen zurückerinnern, werden Sie feststellen, daß wir in jeder Folge einen Aspekt des SAA-Konzepts in den Mittelpunkt gestellt haben.

Galt unser Interesse zunächst den Basismodulen zur Bildschirmausgabe sowie zur Abfrage von Maus und Tastatur, folgte darauf die Pull-Down-Menüverwaltung, die bereits eines der übergeordneten SAA-Module darstellt. Ihr schloß sich in der letzten Ausgabe des *Microsoft System Journals* eine Help-Engine an, die Ihnen die Erstellung und Einbindung von Help-Texten in Ihre Applikationen auf leichte Art und Weise ermöglicht.

Nun also geht es um die SAA-Dialogverwaltung und auch die Module, die ich Ihnen in dieser Folge vorstellen möchte, zählen zu den übergeordneten SAA-Modulen. Wie die Pull-Down-Menüverwaltung und die Help-Engine, stehen auch diese Module ganz im Zeichen größtmöglicher Benutzerfreundlichkeit – einer Forderung, die eines der primären Ziele der SAA-Initiative charakterisiert.

Im Gegensatz zu den vorgenannten Modulen läßt sich eine komplette SAA-Dialogverwaltung jedoch nicht auf den knapp 20 Seiten realisieren, die uns in jeder Folge des System Journal zur Verfügung stehen. Deshalb beschäftigt sich nicht nur diese, sondern auch die kommenden Folgen unserer SAA-Serie mit der SAA-Dialogverwaltung, deren Konzeption – wie Sie gleich feststellen werden – einer Aufteilung in relativ unabhängige Module sogar entgegenkommt.

Was bitte sind Dialogboxen?

Bild 1 zeigt eine typische SAA-Dialogbox, wie man sie in den SAA-Applikationen MS-Works, QuickC 2.0, Quick Pascal oder beispielsweise in der DOS-Shell von DOS 4.0 vorfindet.

Eine Dialogbox erscheint nach der Auswahl eines Kommandos aus der Befehlsleiste als Fenster, das über dem Arbeitsbereich der SAA-Applikation geöffnet wird. Innerhalb dieses Fensters, das von einem einfachen Rahmen umgeben wird, befinden sich die einzelnen Dialog-Felder, auch Dialog-Objekte oder nur Objekte genannt. (Woher der Name stammt, der doch so verdächtig an die im Augenblick heiß diskutierte objektorientierte Programmierweise erinnert, wird am Ende dieses Artikels noch zur Sprache kommen.)

Logisch zusammenhängende Objekte werden dabei räumlich zusammengefaßt und ihre Zusammengehörigkeit nicht selten durch einen sie umgebenden Rahmen gegenüber dem Anwender verdeutlicht. Oft enthält dieser Rahmen einen Titel, der in der Mitte des Rahmens zentriert wird und den Oberbegriff angibt, unter dem die einzelnen Felder stehen.

Der Anwender kann Eingaben innerhalb der einzelnen Objekte in beliebiger Reihenfolge vornehmen, indem er sich mit Hilfe der Tasten Tab und Shift-Tab zwischen den einzelnen Objekten hin- und herbewegt. Das jeweils aktuelle Objekt, also das Objekt, auf das sich seine Eingaben beziehen, wird farblich hervorgehoben und dadurch deutlich sichtbar gemacht.

Aber auch die Maus kann selbstverständlich in die Bearbeitung einer Dialogbox einbezogen werden und oft gestaltet sich die Dateneingabe mit Hilfe der Maus einfacher, als nur mit Hilfe der Tastatur. Die Maus nämlich erlaubt beispielsweise die Anwahl eines Objekts, indem das Objekt einfach angeklickt, der Maus-Cursor also auf das Objekt bewegt und der linke Mausknopf dann kurzzeitig niedergedrückt und wieder losgelassen wird.

Aber auch bei der Dateneingabe innerhalb der einzelnen Objekte vermag die Maus wertvolle Dienste zu leisten, was im folgenden bei der Beschreibung der einzelnen Objekte noch deutlich werden wird.

Darüber hinaus helfen auch die sogenannten »Hotkeys« bei der Auswahl der Objekte weiter. Analog zu den Hotkeys, die im Rahmen der Pull-Down-Menüverwaltung vorgestellt wurden, verfügt jedes Objekt über einen individuellen Hotkey. Die Betätigung des jeweils farblich hervorgehobenen Hotkeys – eines Buchstabens oder einer Zahl – in Verbindung mit der Alt-Taste, führt zur Aktivierung des zugehörigen Objekts.

Vielfalt der Objekte

Wie jede Programmiersprache unterschiedliche Datentypen bereitstellt, um unterschiedliche Arten von Informationen erfassen und verarbeiten zu können, so kennt der SAA-Standard auch verschiedene Arten von Dialog-Objekten, die dem unterschiedlichen Charakter zu verarbeitender Informationen gerecht werden. Und wie jedes Programm eine (nahezu) beliebige Anzahl von Variablen eines bestimmten Typs beinhalten kann, so kann eine Dialog-Box auch (nahezu) beliebig viele verschiedene Objekte ein und desselben Typs aufnehmen.

Als Grundobjekte treten dabei Edit-Felder, Push-Buttons, Radio-Buttons, Auswahl-Boxen und Action-Buttons in Erscheinung, die jeweils unterschiedlichen Aufgaben zugeordnet sind. Wo diese Grundobjekte an ihre Grenze stoßen, können sie leicht erweitert oder gar neue Objekttypen kreiert werden, wobei jedoch gewährleistet sein sollte, daß diese Objekte dem »Look and feel« des SAA-Standards entsprechen, sich also harmonisch in die Reihe der Grundobjekte einfügen.

In der Regel ist die Erarbeitung neuer Objekte jedoch gar nicht notwendig, denn die Grundobjekte decken fast alle Bedürfnisse ab.

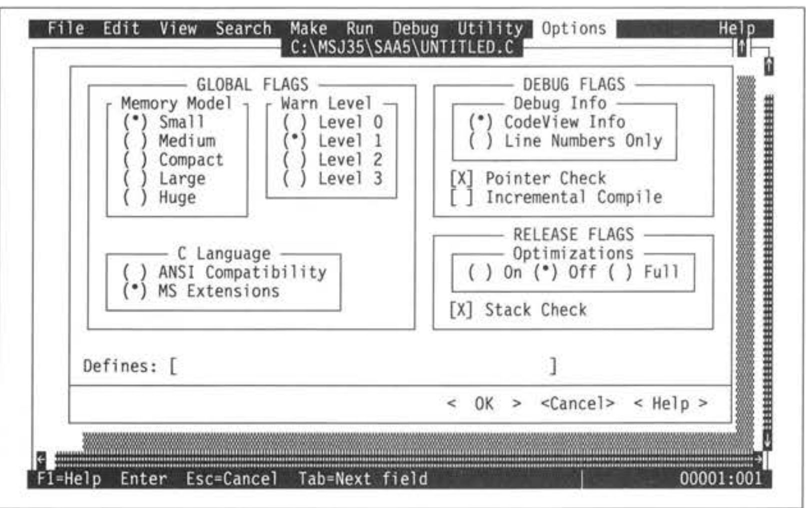


Bild 1: Die Dialogbox zur Einstellung von Compiler-Optionen unter Quick C 2.0 ist eine typische SAA-Dialogbox mit Push-, Radio- und Action-Buttons sowie einem Edit-Feld.

Edit-Felder

So kann das Edit-Objekt beispielsweise für jede Art alphanumerischer Eingaben herangezogen werden. Diese Eingaben werden von Edit in C-Strings abgelegt und können dadurch leicht mit Hilfe der Standard-Funktionen aus der Laufzeitbibliothek des C-Compilers weiterverarbeitet werden.

Edit-Objekte erkennt man an den eckigen Klammern ('[' und ']'), denen in der Regel ein Text vorangeht und die das eigentliche Eingabefeld einschließen. Ist ein Edit-Objekt aktiv, werden diese Klammern und das dazwischen befindliche Eingabefeld farblich hervorgehoben. Der blinkende Cursor markiert die Position, an der das nächste eingegebene Zeichen in das Eingabefeld aufgenommen wird.

Das Erscheinungsbild des Cursors markiert den Eingabemodus. Der normale Cursor, der die untersten beiden Zeilen des unter ihm befindlichen Zeichens überdeckt, zeigt den Überschreib-Modus an, in dem die eingegebenen Zeichen die zuvor eingegebenen Zeichen überschreiben. Im Einfüge-Modus, indem das eingegebene Zeichen an der aktuellen Cursorposition in das Eingabefeld eingeschoben und die nachfolgenden Zeichen nach hinten verschoben werden, überdeckt der Cursor das gesamte unter ihm befindliche Zeichen. Er erscheint dann als Block-Cursor.

Eingefügt werden kann übrigens nur, solange sich an der letzten Position im Eingabefeld ein Leerzeichen befindet, damit keine Zeichen durch das Einfügen aus dem Eingabefeld herausgeschoben werden. Bei der Editierung des Eingabefeldes helfen die gewohnten Editier-Tasten wie Backspace, Delete, Cursor links, Cursor rechts, Home, End und Ctrl-Home.

Das Eingabefeld muß dabei nicht ganz auf dem Bildschirm sichtbar sein. Vielmehr ist es bei langen Eingabefeldern sogar üblich, nur jeweils einen kleinen Teil des Eingabefeldes innerhalb der Dialogbox darzustellen, um Platz zu sparen.

▸▸ *Listing 1: DLG.H*

Mit den Cursor-Tasten kann der Anwender dann den Teil des Eingabefeldes in den Sichtbereich holen, den er bearbeiten möchte. Auf weitere Spezifikationen des Edit-Objekts wird bei der Beschreibung des Moduls DLGEDIT.C, das dieses Objekt realisiert, später eingegangen.

Push-Buttons

Push-Buttons werden überall da eingesetzt, wo die benötigte Information als Ja/Nein-Frage formuliert werden kann. Wenn eine Applikation vom Anwender wissen möchte, ob die Anschrift eines Kunden noch stimmt oder ein Duplikat der Rechnung ausgedruckt werden soll, dann empfehlen sich Push-Buttons, da sie nur zwei Zustände kennen.

Der eigentliche Push-Button wird als ein Zeichen zwischen zwei eckigen Klammern dargestellt, das im Aus-Zustand transparent ist und durch ein Leerzeichen dargestellt wird, während im An-Zustand ein großes X zwischen den eckigen Klammern erscheint. Rechts von diesen Klammern findet sich in der Regel ein Text, der in wenigen Worten die Ja/Nein-Frage formuliert, um deren Beantwortung gebeten wird.

Ist ein Push-Button als aktives Objekt markiert worden, kann die Umschaltung zwischen den beiden Zuständen mit Hilfe der Leertaste erfolgen. Alternativ dazu genügt es auch, den Push-Button mit der Maus anzuklicken.

Radio-Buttons

Anders als ein Push-Button, der immer für sich allein steht, bestehen Radio-Buttons grundsätzlich aus einer Gruppe von Buttons (Knöpfen) von denen jeweils nur einer ausgewählt sein kann. Ihren Namen verdanken sie den Schaltern an einem Radio, die die Auswahl eines Frequenz-Bandes erlauben. Hier – und dies ist charakteristisch für Radio-Buttons – ist die Auswahl von Lang-, oder Kurz-, oder Mittelwelle möglich, es kann aber immer nur genau *eine* Einstellung ausgewählt werden. Wird ein Knopf niedergedrückt, so springt gleichzeitig der bisher niedergedrückte Knopf wieder heraus.

Um eine optische Unterscheidung zwischen Radio- und Push-Buttons zu gewährleisten, werden Push-Buttons nicht von eckigen, sondern von runden Klammern umgeben. Ist ein Push-Button niedergedrückt, wird zwischen diesen Klammern das Zeichen »•« angezeigt. Ansonsten ist der Platz zwischen den beiden Klammern leer.

Zusammengehörende Push-Buttons werden grundsätzlich untereinander aufgeführt, wobei jedem Button ein Text folgt, der die jeweilige Einstellung nennt. Zwar handelt es sich bei Push-Buttons, die aus mehreren Knöpfen bestehen, um ein Objekt, doch kann zwischen den einzelnen

```
/*
[ ]──────────────────────────────────────────────────────────────[ ]
 |                                                                |
 |     Include-Datei    : DLG.H                                   |
 |     zur Einbindung   : der Funktionen zur Erstellung und Verwaltung von
 |                        SAA-Dialogboxen                         |
 |     erstellt am      : 15.06.1989                              |
 |     letztes Update am: 20.06.1989                              |
 |                                                                |
 |     (Copyright)      : 1989 by MICHAEL TISCHER                 |
 |                                                                |
[ ]──────────────────────────────────────────────────────────────[ ]
*/

#include "vio.h"               /* diese beiden SAA-Include-Dateien aufnehmen */
#include "kbm.h"

/*— Makros ──────────────────────────────────────────────────────────────*/

#define F(farbe) ( dlgcol.farbe )  /* liefert Farbe aus der DLGCOL-Struktur */

/*— die folgenden Makros helfen bei der Anlage von DLGDATEN-Strukturen für */
/*— die einzelnen Objekte innerhalb einer Dialogbox                        */

#define EDIT( p )   { &p, &std_edit_fkt }
#define ACTBUT( p ) { &p, &std_ab_fkt   }

/*— Konstanten ──────────────────────────────────────────────────────────*/

#define HOTKEY     '#'                /* Zeichen, das einen Hotkey markiert */
#define NOKEY      ((TASTE) 0)                                /* keine Taste */
#define NOPAINTFKT ((PAINTFKT) 0)                   /* keine 'Mal'-Funktion */

/*— Rückgabecodes für die TASTFKT ───────────────────────────────────────*/

#define TF_MAUS       250 /* aktiviere mich aufgrund eines Maus-Ereignisses */
#define TF_AKTIV      251 /* aktiviere mich aufgrund einer Taste            */
#define TF_WEITER     252 /* Taste/Mausereignis wurde nicht verarbeitet     */
#define TF_ACCEPTED   253 /* Taste/Mausereignis wurde verarbeitet           */
#define TF_FELD_VOR   254 /* ein Feld weiterschalten                        */
#define TF_FELD_RUECK 255 /* ein Feld zurück                                */
                          /* alle weiteren Codes werden als Terminations-   */
                          /* codes aufgefaßt                                */

/*— Typdeklarationen ────────────────────────────────────────────────────*/

typedef BYTE BOOL;

typedef struct   /* Farben, die innerhalb einer Dialogbox Verwendung finden */
 {
  BYTE dlgbox,              /* Füll-Farbe für gesamte Dialogbox und Rahmen */
       nm,                                              /* normale Farbe */
       hi,                                        /* hervorgehobene Farbe */
       hk,                                        /* Farbe für den Hotkey */
       da,                        /* Farbe für inaktive Felder (disabled) */
       dk;                    /* Farbe für den Hotkey in inaktiven Feldern */
 } DLGCOL;

extern DLGCOL dlgcol;                /* Variable mit Farben für Dialog-Boxen */


/*— Deklaration der Funktionen, die jedes Dialog-Feld bereitstellen muß ───
 |                                                                        |
 |    STARTFKT  : Wird beim Aufbau der Maske aufgerufen. Liefert dem Aufru-
 |                fer einen Zeiger zurück, der bei allen kommenden Aufrufen
 |                der verschiedenen Dialogfunktionen übergeben wird.      |
 |                                                                        |
 |    Jeder der folgenden Funktionen wird bei ihrem Aufruf der Zeiger über-
 |    geben, den die STARTFKT zurückgeliefert hat und an der die Funktion |
 |    feststellen kann, welches der möglicherweise mehreren Dialog-Felder |
 |    dieses Typs angesprochen wird.                                      |
 |                                                                        |
 |      AKTIVFKT  : Aktiviert ein Dialog-Feld, nachdem es zuvor über die  |
 |                  Funktion CANAKTFKT seine Bereitschaft erklärt hat,    |
 |                  aktiviert zu werden. Teilt dem Dialog-Feld anhand eines
 |                  der TF_...-Konstanten mit, warum es aktiviert wurde.  |
 |                                                                        |
 |      TASTFKT   : Der Aufruf erfolgt sowohl während das Dialog-Feld aktiv
 |                  ist, manchmal aber auch, wenn ein anderes Dialog-Feld |
 |                  aktiv ist, dieses die Taste aber abgelegt hat. In diesem
 |                  Fall muß das Dialog-Feld feststellen, ob es die Taste |
 |                  verarbeiten kann - ggf. sogar aktiviert wird.         |
 |                                                                        |
 |      MAUSFKT   : Wie TASTFKT wird diese Funktion sowohl aufgerufen, wäh-
 |                  rend ein Dialogfeld aktiv ist, als auch, wenn es nicht|
 |                  aktiv ist, um festzustellen, ob es durch ein Mausereig-
 |                  nis aktiviert werden muß.                             |
 |                                                                        |
 |      CANAKTFKT : Teilt dem Aufrufer mit, ob das Dialog-Feld bereits ist,
 |                  aktiviert zu werden.                                  |
 |                                                                        |
 |      NEWVALFKT : Wird nicht vom Scheduler, sondern von einer Verbindungs-
 |                  prozedur des Anwenders/Programmierers aufgerufen, um dem
 |                  Dialog-Feld mitzuteilen, daß sich sein Inhalt durch ein
 |                  äußeres Ereignis verändert hat. In den Standardversionen
 |                  der verschiedenen Dialogtypen sind diese Funktionen   |
 |                  Dummys.                                               |
 |                                                                        |
 |      DEAKFKT   : Teilt dem Dialog-Feld mit, daß es daktiviert wurde.   |
 |                                                                        |
 |      ENDFKT    : Wird nach der Beendigung der Eingabe innerhalb der Dia-
 |                  logbox aufgerufen, damit das Dialog-Feld einen Reset auf
 |                  seine Daten durchführen kann. Vom Bildschirm muß es sich
 |                  nicht entfernen.                                      |
 |                                                                        |
 ──────────────────────────────────────────────────────────────────────────*/

typedef void * (*STARTFKT) ( void * iptr );
typedef void   (*NEWVALFKT)( void * iptr, void * dptr );
typedef void   (*ENDEFKT)  ( void * iptr );
typedef void   (*DEAKFKT)  ( void * iptr );
typedef void   (*AKTIVFKT) ( void * iptr, BYTE why );
typedef BYTE   (*TASTFKT)  ( void * iptr, TASTE key );
typedef BYTE   (*MAUSFKT)  ( void * iptr, BYTE x, BYTE y, BYTE ev );
typedef BOOL   (*CANAKTFKT)( void * iptr );

typedef struct          /* Struktur mit den verschiedenen Funktions-Pointern */
 {
  STARTFKT  start;
  NEWVALFKT newval;
  ENDEFKT   ende;
  TASTFKT   taste;
  DEAKFKT   deak;
  MAUSFKT   maus;
  AKTIVFKT  aktiv;
  CANAKTFKT can;
 } DLGFKT;

typedef DLGFKT *DLGFKTPTR;     /* Zeiger auf eine Dialog-Funktions-Struktur */
/*— Markos, die bei der Definition von Eingabemengen für EDIT-Felder helfen*/
```

```c
#define SM(m, x, y )        EdSetMenge(m, x, y, TRUE )
#define CM(m, x, y )        EdSetMenge(m, x, y, FALSE)
#define ClearMenge(m)       CM( m, 0, 255 )
#define SetMengeAll(m)      SM( m, 0, 255 )
#define SetMengeInt(m)      ( SM( m, '0', '9' ), SM( m, '+', '-' ) )

#define SetMengeAN(m)       ( SM( m, ' ', '~' ), SM( m, 'ü', 'ü' ),   \
                              SM( m, 'ä', 'ä' ), SM( m, 'ö', 'ö' ),   \
                              SM( m, 'Ü', 'Ü' ), SM( m, 'Ö', 'Ö' ),   \
                              SM( m, 'Ä', 'Ä' ), SM( m, 'ß', 'ß' ) )

#define SetMengeDatei(m)    ( SM( m, 'a', 'z' ), SM( m, 'A', 'Z' ),   \
                              SM( m, '!', '%' ), CM( m, 34, 34 ),     \
                              SM( m, '-', '-' ), SM( m, '_', '_' ),   \
                              SM( m, 'ä', 'ä' ), SM( m, 'ö', 'ö' ),   \
                              SM( m, 'Ü', 'Ü' ), SM( m, 'Ö', 'Ö' ),   \
                              SM( m, 'Ä', 'Ä' ), SM( m, 'ß', 'ß' ) )

#define SetMengePath(m)     ( SetMengeDatei(m), \
                              SM( m, ':', ':' ), SM( m, 92, 92 ) )

/*— Beschreibung der Struktur eines Dialogbox-Beschreibers ————————*/

extern DLGFKT std_edit_fkt;          /* Funktionen zur Verwaltung von Edit- */
extern DLGFKT std_ab_fkt;            /* Feldern und Action Buttons          */

typedef void (*PAINTFKT)( void );    /* Fkt. zur Gestaltung einer Dialogbox */

typedef struct            /* Daten, die für jedes Dialog-Feld benötigt werden */
 {
  void      * daten;                                /* Zeiger auf Datenblock */
  DLGFKTPTR   fkts;            /* Zeiger auf Struktur mit Dialog-Funktionen */
 } DLGDATEN;

typedef DLGDATEN * DLGDPTR;                         /* Zeiger auf Dialog-Daten */

typedef struct                                    /* beschreibt eine Dialogbox */
 {
  BYTE     x_start,                           /* Koordinaten obere linke Ecke */
           y_start,                           /* der Dialogbox                */
           x_len,                                 /* Größe der Dialogbox in */
           y_len,                                 /* Spalten und Zeilen     */
           anz;                                  /* Anzahl der Dialogfelder */
  PAINTFKT paintfkt;                /* Funktion zur Gestaltung der Dialogbox */
  DLGDPTR  datptr;                          /* Zeiger auf Vektor mit Zeigern */
                                            /* auf Dialogfeld-Daten          */
 } DIGDES;

typedef DIGDES * DIGDESPTR;                 /* Zeiger auf Dialogbox-Struktur */

/*— Beschreibung der Strukturen, die für ein alphanumerisches Eingabefeld -*/
/*— vom Type EDIT benötigt werden                                          -*/

typedef BYTE EMENGE[ 64 ];                                  /* Eingabemenge */
typedef BYTE *EMP;                           /* Zeiger auf eine Eingabemenge */

typedef struct                 /* beschreibt ein alphanumerisches Eingabefeld */
 {
  BOOL   enabled;                         /* darf das Feld angewählt werden? */
  BYTE   x,                          /* Anfangskoordinate relativ zur oberen */
         y,                          /* linken Ecke des Dialogbox            */
         len,                          /* Anzahl der Zeichen im Eingabefeld */
         visi;                            /* Anzahl der sichtbaren Zeichen */
  char * text;                                /* Text vor dem Eingabefeld */
  BYTE * eingabe;                           /* Zeiger auf den Eingabepuffer */
  EMP    mptr;                     /* Zeiger auf die Menge erlaubter Zeichen */
 } EDITFELD;

/*— Beschreibung der Strukturen, die für die Verwaltung von Action Buttons */
/*— (AB) benötigt werden.                                                  */
/*— ACHTUNG! Action Buttons darf es im Gegensatz zu EDIT-Feldern, Push-    */
/*— Buttons etc. nur einmal pro Dialogbox geben                            */

typedef struct                               /* beschreibt einen Action-Button */
 {
  BOOL   enabled;                           /* darf der AB ausgewählt werden? */
  BYTE   x,                          /* Bildschirmposition relativ zur oberen */
         y;                          /* linken Ecke des Dialogfensters       */
  char * name;                                /* Pointer auf String mit Text */
  TASTE  key;             /* Taste, mit der dieser AP (zusätzlich zum evtl. */
                          /* vorhandenen Hotkey) verbunden wird              */
 } ONEAB;

typedef ONEAB *ABPTR;                          /* Zeiger auf einen AB-Beschreiber */

typedef struct                     /* beschreibt eine Gruppe von Action-Buttons */
 {
  BYTE anz,                                      /* Anzahl der Action-Buttons */
       standard;                                   /* Default Action-Button */
  ABPTR abp;                             /* Zeiger auf Vektor mi AB-Beschreibern */
 } ABGROUP;

/*— öffentliche Funktionen ———————————————————————————————*/

BYTE  DlgStart  ( DIGDESPTR dptr );
TASTE DlgPrint  ( BYTE x, BYTE y, char * str, BYTE fn, BYTE fh );
void  DlgDelay  ( int pauslen );
void  DlgBox    ( BYTE x, BYTE y, BYTE xlen, BYTE ylen,
                  BYTE typ, char * titel, BOOL aktiv );
void  EdSetMenge( EMP mptr, BYTE von, BYTE bis, BOOL set );

/*— Funktionsdeklarationen über Makros ——————————————————————*/

#define MausPause()  DlgDelay( 1 )                 /* Pause bei Mausereignissen */
```

◄◄ *Listing 1: Ende*

Push-Buttons mit Tab und Shift-Tab hin- und hergesprungen werden, so als ob es sich um unterschiedliche Objekte handelte. Die Auswahl eines von mehreren Radio-Buttons erfolgt analog zu den Push-Buttons mit Hilfe der Leertaste oder durch Anklicken mit der Maus.

Auswahl-Boxen

Überall da, wo zwar auch nur eine aus mehreren Möglichkeiten ausgewählt werden kann, diese Möglichkeiten zum Zeitpunkt der Kompilation aber noch nicht feststehen oder groß an der Zahl sind, werden Auswahl-Boxen eingesetzt. Umgeben von einem Rahmen und am rechten oder unteren Rand mit einem sogenannten Slider versehen, enthält eine Auswahl-Box eine unterschiedliche Anzahl gleich breiter Auswahlen, von denen immer nur so viele gleichzeitig zu sehen sind, wie in die Auswahl-Box am Bildschirm passen.

In der Art einer Tabellenkalkulation können weitere Einträge mit Hilfe der Cursor-Tasten in die Auswahl-Box geholt werden, doch werden dafür andere Einträge aus dem Sichtbereich herausgescrollt. Man muß sich diese Organisation der Einträge als eine Art Matrix vorstellen, von der immer nur ein bestimmter Teil im Sichtbereich der Auswahl-Box angezeigt wird.

In welcher Scroll-Richtung darüber hinaus noch weitere Einträge existieren, zeigt jeweils ein Slider an, der auch zur Fortbewegung zwischen den einzelnen Einträgen mit Hilfe der Maus dient. Dem Programmierer steht es bei der Deklaration einer Auswahl-Box frei, die Breite der einzelnen Einträge sowie die Anzahl der untereinander oder nebeneinander angezeigten Einträge zu wählen.

Die in *Bild 2* dargestellte Dateiauswahlbox beinhaltet gleich zwei dieser Auswahl-Boxen. Eine, um ein Laufwerk bzw. Verzeichnis auszuwählen und eine andere, um eine der darin befindlichen Dateien zu markieren.

Action-Buttons

Den letzten Grundtyp unter den verschiedenen Dialog-Objekten stellen die Action-Buttons dar, die sich von den bisher beschriebenen Objekt-Typen insofern unterscheiden, als das jedes Dialogfeld über genau ein Objekt vom Typ Action-Button verfügen *muß*.

Daß dem so ist, hängt mit dem besonderen Charakter der Action-Buttons zusammen, denn nur mit ihrer Hilfe kann die Eingabe in der Dialogbox beendet werden.

Dabei kann ein Action-Button aus einer beliebigen Anzahl verschiedener Knöpfe bestehen, die jeweils mit einer bestimmten Aktion verbunden werden. Zumindest drei dieser Knöpfe findet man denn auch in fast jeder Dialogbox:

• einer mit dem Namen »OK«, der zum Schließen der Dialogbox führt und die eingegebenen Daten übernimmt,

• einer, der den Titel »Abbruch« trägt und ebenfalls zum Schließen der Dialogbox führt, allerdings ohne daß die Daten übernommen werden, und schließlich

• einer, mit dem Namen »Hilfe«, der Hilfsinformationen über die Dialogbox auf den Bildschirm holt und nach deren Betrachtung wieder in die Dialogbox zurückkehrt.

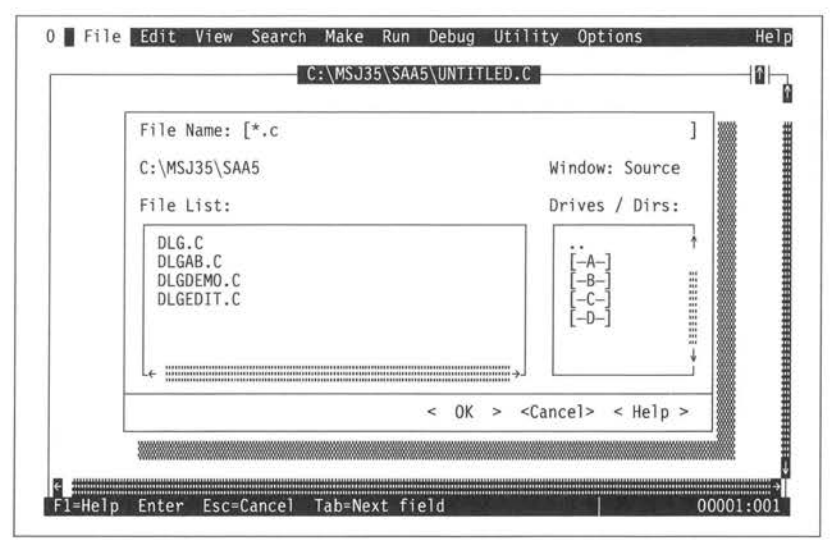


Bild 2: Die Dateiauswahlbox von Quick C enthält gleich zwei Auswahl-Boxen und ein Edit-Feld.

Erkennen können Sie Action-Buttons an den Zeichen »<« und »>«, zwischen denen der Name des Buttons aufgeführt wird. Plaziert werden sie in der Regel in der untersten Zeile der Dialogbox – von den anderen Objekten durch eine horizontale Linie getrennt.

Jeder der verschiedenen Buttons kann mit einer bestimmten Taste verbunden werden, so daß beispielsweise die Betätigung der Esc-Taste der Auswahl des »Abbruch«-Buttons gleichkommt.

Auch gibt es einen Standard-Button, der immer dann aktiv ist, wenn sich der Anwender nicht innerhalb der Gruppe der Action-Buttons befindet, also nicht im Begriff ist, einen bestimmten Button auszuwählen. Sofern kein anderer Button mit der Return-Taste verbunden ist, wird bei der Betätigung dieser Taste der Standard-Button ausgelöst.

Interdependenzen in Dialogboxen

Dialogboxen, wie wir sie in *Bild 1* sehen, sind sehr einfach zu verwirklichen, da hier keine Interdependenzen, keine Abhängigkeiten, zwischen den einzelnen Feldern bestehen. Jedes nimmt eine Information auf, die völlig isoliert von den Eingaben in andere Felder betrachtet werden kann.

Oft ist es jedoch gerade nicht so. Oft gibt es Abhängigkeiten zwischen den verschiedenen Objekten, wie z.B. innerhalb der Dateiauswahlbox in *Bild 2*. Wählt der Anwender eine Dateikennung im Edit-Feld aus, muß die Dateibox neu aufgearbeitet werden, um alle Dateien aufzuführen, die auf diese Dateikennung passen. Dieser Vorgang muß wiederholt werden, sobald in der Auswahl-Box, die der Auswahl des Laufwerks gewidmet ist, ein neues Laufwerk ausgewählt wird. Andersherum muß der Dateiname im Edit-Feld verändert werden, sobald der Anwender innerhalb der Dateibox einen neuen Dateinamen anwählt.

Da die Objekte innerhalb einer Dialogbox auf fast beliebige Art und Weise miteinander verknüpft und diese Verknüpfungen zum Teil aufwendige Operationen (etwa die Suche nach passenden Dateinamen in einem Verzeichnis) beinhalten können, ist es dem Dialog-Manager natürlich unmöglich, diese Verknüpfungsoperationen als Teil seiner Aufgaben wahrzunehmen.

Da auf diese Verknüpfungen aber auch nicht verzichtet werden kann, stellt sich an den Dialog-Manager (auch Scheduler genannt) die Forderung, daß Funktionen des Programmierers Zugriffsmöglichkeiten auf den Prozeß der Dateneingabe innerhalb der Dialogbox einzuräumen ist. Diese Forderung bestimmt in hohem Maße die Struktur des Dialog-Managers, wie sie im folgenden Abschnitt beschrieben wird.

Die Struktur des Dialog-Managers

Der Dialog-Manager, der zusammen mit seinen Hilfsfunktionen in der Datei DLG.C untergebracht ist, trägt den Namen DlgStart(). Er agiert unabhängig von der Art der Objekte innerhalb der Dialogbox, denn ihm gegenüber treten die einzelnen Objekte nur in Form von acht sogenannten Dialog-Funktionen auf, die jedes Objekt unterstützten, besser gesagt, beinhalten muß. Diese Funktionen bzw. Zeiger, die auf sie verweisen, werden innerhalb der Include-Datei DLG.H definiert, die jede Applikation einbinden muß, die sich der Dienste des Dialog-Managers bedienen möchte.

DlgStart() muß bei seinem Aufruf ein Zeiger auf eine Struktur vom Typ DIGDES übergeben werden, die alle Informationen enthält, die der Dialog-Manager zur Verwaltung der Dialogbox benötigt.

Dies ist zunächst einmal die Koordinate der oberen linken Ecke der Dialogbox sowie die Größe in Zeilen und Spalten. Mit Hilfe dieser Information öffnet der Dialog-Manager ein Bildschirmfenster, löscht seinen Inhalt und zieht einen einfachen Rahmen darum.

Des weiteren findet sich in dieser Struktur die Anzahl der Objekte und ein Zeiger mit dem Namen paintfkt, der auf eine Funktion des Aufrufers verweisen kann, die während des Aufbaus der Dialogbox vom Dialog-Manager aufgerufen wird. Dadurch soll dem Aufrufer die Möglichkeit geboten werden, in die Gestaltung der Dialogbox einzugreifen, etwa, um eine waagerechte Linie zwischen den Action-Buttons und den anderen Objekten zu ziehen oder einen Text in die Dialogbox zu schreiben. Möchte der Aufrufer von dieser Möglichkeit keinen Gebrauch machen, kann er an der entsprechenden Stelle innerhalb der Struktur an Stelle eines Funktionsnamens die Konstante NOPAINTFKT einsetzen.

◄◄ *Listing 2: DLG.C*

```
/*************************************************************************/
/*                              D L G . C                                */
/*-----------------------------------------------------------------------*/
/*    Aufgabe        : Stellt im Rahmen der SAA-Serie eine Funktion zur  */
/*                     Erstellung und Verwaltung beliebiger Dialogboxen zur */
/*                     Verfügung.                                        */
/*-----------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                   */
/*    entwickelt am  : 13.07.1989                                        */
/*    letztes Update : 17.07.1989                                        */
/*-----------------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] DLG.C [ DLGEDIT.C, DLGAB.C ...] /C */
/*                     dann mit einem anderen Modul linken               */
/*************************************************************************/

/*— Include-Dateien einbinden ------------------------------------------*/
#include <string.h>
#include <memory.h>
#include <malloc.h>
#include <dos.h>
#include "dlg.h"

/*— globale Variablen --------------------------------------------------*/

DLGCOL dlgcol = { 0x70, 0x70, 0x07, 0x01, 0x70, 0x70 };    /* Dialog-Farben */

BOOL      ende,                               /* zeigt das Ende der Eingabe an */
          maus;               /* ist TRUE, wenn die Mausaktivität verfolgt wird */
BYTE      aktiv,                            /* Nummer des aktiven Dialog-Feldes */
          retcode;                                /* Return-Wert für Aufrufer */
void   ** idptr,                /* Zeiger auf Vektor mit Id-Zeigern der Felder */
        * aktip;                      /* Zeiger auf Element des aktiven Feldes */
DLGDPTR   aktdlgp;              /* Zeiger auf Infos über aktuelles Dialog-Feld */
DIGDES    dp;                                     /* Daten der Dialogstruktur */

#define FKT(x)  (*(aktdlgp->fkts->x))      /* Dlg-Fkt. des akt. Feldes aufr. */
#define FKTL(x) (*(dlgdptr->fkts->x))

/************************************************************************
* Funktion          : D l g i P r o c e s s K e y                        *
**---------------------------------------------------------------------**
* Aufgabe           : Interne Funktion des DLG-Moduls, bearbeitet eine ein- *
*                     gegebene Taste.                                    *
* Eingabe-Parameter: KEY = Code der betätigten Taste                     *
* Return-Wert       : keiner                                             *
* Globals           : ENDE, RETCODE, AKTIV, AKTIP, AKTDLGP               *
*************************************************************************/
BYTE DlgiProcessKey( TASTE key )
{
 BYTE      neuaktiv;                          /* Nummer des neuen aktiven Feldes */
 void   ** lidptr;                                 /* Laufzeiger in IDPTR-Vektor */
 DLGDPTR   dlgdptr;             /* Laufzeiger in Vektor auf den dp.datptr zeigt */

 switch ( retcode = FKT(taste)( aktip, key ) ) /* Taste an akt. Feld senden */
  {
   case TF_FELD_VOR   :              /* nachfolgendes Dialog-Feld aktivieren */
    neuaktiv = aktiv;                                     /* Suche im nächsten */
    lidptr = idptr + neuaktiv;                            /* Dialog-Feld be-   */
    dlgdptr = aktdlgp;                                    /* ginnen            */
    do                                                        /* Suchschleife */
     {
     if ( ++neuaktiv == dp.anz )                  /* zum ersten Feld springen? */
      {                                                                 /* Ja */
       neuaktiv = 0;                                  /* bei Feld #0 fortfahren */
       lidptr   = idptr;                       /* Zeiger auf das jeweils erste */
       dlgdptr  = dp.datptr;                   /* Element im Vektor setzen     */
      }
     else                                       /* Nein, es folgt noch ein Feld */
      {
       ++lidptr;                     /* die beiden Zeiger auf das nächste  '*/
       ++dlgdptr;                    /* Element im jeweiligen Vektor setzen */
      }
     }
    while ( ! ( neuaktiv == aktiv  ||  FKTL(can)( *lidptr ) ) );
    if ( neuaktiv != aktiv )             /* wurde ein anderes Feld aktiviert? */
     {                                                                  /* Ja */
      FKT(deak)( aktip );                         /* aktuelles Feld deaktivieren */
      aktip   = *lidptr;    /* Ptr auf Datenblock des aktiven Feldes merken */
      aktdlgp = dlgdptr;   /* Ptr auf Daten mit Infos über akt. Feld merken */
      aktiv   = neuaktiv;                           /* Nummer des Feldes merken */
      FKT(aktiv)( aktip, TF_FELD_VOR );               /* neues Feld aktivieren */
     }
    break;

   case TF_FELD_RUECK :             /* vorhergehendes Dialog-Feld aktivieren */
    neuaktiv = aktiv;                                     /* Suche im vorher- */
    lidptr = idptr + neuaktiv;                            /* gehenden Dialog- */
    dlgdptr = aktdlgp;                                    /* Feld beginnen    */
    do                                                        /* Suchschleife */
     {
     if ( neuaktiv— == 0)                        /* zum letzten Feld springen? */
      {                                                                 /* Ja */
       neuaktiv = dp.anz-1;                       /* bei Feld #anz-1 fortfahren */
       lidptr   = idptr + neuaktiv;          /* Zeiger auf das jeweils letzte */
       dlgdptr  = dp.datptr + neuaktiv;      /* Element im Vektor setzen      */
      }
     else                                   /* Nein, es geht noch ein Feld voraus */
      {
       —lidptr;                   /* die beiden Zeiger auf das vorhergehende */
       —dlgdptr;                  /* Element im jeweiligen Vektor setzen     */
      }
     }
    while ( ! ( neuaktiv == aktiv  ||  FKTL(can)( *lidptr ) ) );
    if ( neuaktiv != aktiv )             /* wurde ein anderes Feld aktiviert? */
     {                                                                  /* Ja */
      FKT(deak)( aktip );                         /* aktuelles Feld deaktivieren */
      aktip   = *lidptr;    /* Ptr auf Datenblock des aktiven Feldes merken */
      aktdlgp = dlgdptr;   /* Ptr auf Daten mit Infos über akt. Feld merken */
      aktiv   = neuaktiv;                           /* Nummer des Feldes merken */
      FKT(aktiv)( aktip, TF_FELD_RUECK );             /* neues Feld aktivieren */
     }
    break;

   case TF_WEITER     :         /* Feld suchen, das diese Taste verarbeitet */
    neuaktiv = aktiv;                                     /* Suche im nächsten */
    lidptr = idptr + neuaktiv;                            /* Dialog-Feld be-   */
    dlgdptr = aktdlgp;                                    /* ginnen            */
    do                                                        /* Suchschleife */
     {
     if ( ++neuaktiv == dp.anz )                  /* zum ersten Feld springen? */
      {                                                                 /* Ja */
       neuaktiv = 0;                                  /* bei Feld #0 fortfahren */
       lidptr   = idptr;                       /* Zeiger auf das jeweils erste */
       dlgdptr  = dp.datptr;                   /* Element im Vektor setzen     */
      }
     else                                       /* Nein, es folgt noch ein Feld */
      {
       ++lidptr;                     /* die beiden Zeiger auf das nächste   */
       ++dlgdptr;                    /* Element im jeweiligen Vektor setzen */
      }
     }
```

Als letzten Eintrag enthält eine DIGDES-Struktur einen Zeiger auf einen Vektor, mit einem Eintrag für jedes der Objekte innerhalb der Dialogbox. Solch ein Vektor setzt sich aus Elementen vom Typ DLGDATEN zusammen. Dieser Typ stellt wiederum eine Struktur dar, die zwei Zeiger enthält. Der erste ist ein void-Zeiger, der auf eine beliebige Struktur verweisen kann, die Daten über das jeweilige Objekt enthält. Dem Dialog-Manager ist die Struktur dieser Daten unbekannt, doch ist sie für ihn auch gar nicht von Interesse, da er sie selbst nicht verarbeiten muß, sondern diesen Zeiger lediglich an die Dialog-Funktionen weitergibt, die mit ihm ganz nach Belieben verfahren können.

Der zweite Zeiger hingegen ist typisiert – er zeigt auf eine Struktur vom Typ DLGFKT, die Zeiger auf die acht Dialog-Funktionen eines Objekts beinhaltet. Daß die einzelnen Funktionszeiger hier nicht direkt aufgeführt, sondern durch einen Zeiger auf eine DLGFKT-Struktur referenziert werden, hat einen ganz besonderen Grund. So lange Sie nämlich keine Abhängigkeiten zwischen den einzelnen Objekten definieren müssen und daher mit den Standard-Objekten arbeiten können, werden auch immer die Standard-Dialog-Funktionen des jeweiligen Objekts aufgerufen. Es ist deshalb einfacher, einmal eine Struktur mit den Zeigern auf diese Funktionen als globale Variable zu definieren und sie innerhalb der DLGDATEN-Struktur zu referenzieren, als die einzelnen Funktionszeiger in jeder dieser Strukturen separat aufzuführen und so nicht nur Speicherplatz, sondern auch wertvolle Zeit beim Eintippen zu vergeuden.

Darüber hinaus stellt jedes Objekt ein Makro zur Verfügung, mit dessen Hilfe Sie eine DLGDATEN-Struktur für Objekte dieses Typs erstellen können, ohne mit dem Zeiger auf die DLGFKT-Struktur in Berührung zu kommen. Das folgende Beispiel zeigt dieses Makro für EDIT-Felder.

```
#define EDIT( p ) { &p, &std_edit_fkt }
```

Hier wird neben dem Zeiger auf die Daten also auch direkt einer auf die Struktur std_edit_fkt referenziert, die vom Modul DLGEDIT.C bereitgestellt wird und Zeiger auf die Standard-Dialog-Funktionen für ein Edit-Objekt enthält. Die Reihenfolge, in der die einzelnen Objekte innerhalb des Vektors mit den DLGDATEN-Strukturen aufgeführt werden, bestimmt auch die Reihenfolge, in der die einzelnen Felder mit Hilfe von Tab und Shift-Tab angesprungen werden können.

Dialog-Funktionen zur Verbindung zwischen Dialog-Manager und einem Objekt

Die erste der acht Dialog-Funktionen, die START-FKT, ruft der Dialog-Manager nur einmal wäh-

▸▸ Listing 2: Fortsetzung

rend der Initialisierung der Dialogbox, unmittelbar nach dem Öffnen des Dialogfensters, auf. Damit sie auf die Daten des zugehörigen Objekts zugreifen kann, wird ihr bei ihrem Aufruf auch gleich der Zeiger auf die Objekt-Daten aus der DLGDATEN-Struktur des Objekts übergeben.

Als Return-Wert erwartet der Dialog-Manager einen Zeiger vom Typ void, den er bei allen künftigen Aufrufen der anderen Dialog-Funktionen übergibt. Um die Bedeutung dieses Zeigers zu ermessen, müssen Sie sich in Erinnerung rufen, daß jedes Objekt (ausgenommen die Action-Buttons) innerhalb einer Dialog-Box mehrmals auftreten kann. Da aber zumindest die Standard-Objekte in jeder DLGDATEN-Struktur eines Objekts auf den gleichen Satz von Dialog-Funktionen verweisen, werden ein- und dieselben Funktionen für mehrere Objekte in unvorhersehbarer Reihenfolge aufgerufen.

Wenn für jedes Objekt eines bestimmten Objekttyps (für jede »Instanz« eines Objekts) nun interne Informationen, wie etwa die Cursor-Position innerhalb eines Eingabefelds oder der Status eines Push-Buttons verwaltet werden müssen, dann müssen die einzelnen Dialog-Funktionen die verschiedenen Instanzen auseinanderhalten können. Deshalb deklarieren die Dialog-Funktionen von Objekt-Typen, die interne Informationen über ihr Objekt verwalten müssen, in der Regel eine interne Struktur, in der alle diese Informationen Platz finden. Beim Aufruf der STARTFKT allokieren sie auf dem Heap Speicherplatz für eine solche Struktur, initialisieren sie und liefern die Adresse dieser Struktur als Zeiger an die STARTFKT zurück. (Neben internen Daten sollte auch die Adresse der Objekt-Daten in dieser Struktur verzeichnet sein, damit jede Dialog-Funktion neben den internen Daten auch auf die Objekt-Daten zugreifen kann.)

Indem der Dialog-Manager allen anderen Dialog-Funktionen diesen Zeiger übergibt und die Dialog-Funktionen auf die Daten innerhalb der dadurch referenzierten Struktur zugreifen, können mit einem Satz von Dialog-Funktionen eine beliebige Anzahl von Instanzen eines Objekt-Typs gleichzeitig bearbeitet werden. So weit zur STARTFKT.

Die CANFKT ruft der Dialog-Manager immer dann auf, wenn er nach einem Objekt sucht, das bereit ist, als aktives Objekt ausgewählt zu werden. Ist diese Bereitschaft vorhanden, sollte die CANFKT TRUE zurückliefern, anderenfalls FALSE.

Immer, wenn der Dialog-Manager ein Objekt aktivieren möchte, also beispielsweise, wenn ein vorhergehender Aufruf der CANFKT eines Objekts den Wert TRUE zurückgeliefert hat, ruft der Dialog-Manager die AKTIVFKT auf. Ihr Aufruf teilt dem Objekt mit, daß und warum das Objekt aktiviert wurde. Dadurch wird dem Objekt die Möglichkeit geboten, sich selbst auf dem Bildschirm hervorzuheben, den Cursor zu plazieren

```
   while ( neuaktiv != aktiv  &&
          ( retcode = FKTL(taste)(*lidptr, key) ) == TF_WEITER );
   if ( neuaktiv != aktiv )                     /* wurde die Taste verarbeitet? */
    {                                                                  /* Ja */
     if ( retcode == TF_AKTIV )                     /* das Feld aktivieren? */
      {                                                                /* Ja */
       FKT(deak)( aktip );                      /* aktuelles Feld deaktivieren */
       aktip   = *lidptr;  /* Ptr auf Datenblock des aktiven Feldes merken */
       aktdlgp = dlgdptr;   /*Ptr auf Daten mit Infos über akt. Feld merken*/
       aktiv   = neuaktiv;                        /* Nummer des Feldes merken */
       FKT(aktiv)( aktip, TF_AKTIV );                /* neues Feld aktivieren */
      }
     else                  /* Nein, mit zurückgeliefertem Code terminieren */
      ende = TRUE;                                      /* Return-Flag setzen */
    }
   break;
  case TF_ACCEPTED   :                             /* Taste wurde verarbeitet */
   break;                                                  /* nichts passiert */
  default            :            /* jeder andere Code bedeutet Termination */
   FKT(deak)( aktip );                         /* aktuelles Feld deaktivieren */
   ende = TRUE;                                         /* Return-Flag setzen */
   break;
  }
}

/***************************************************************************
*  Funktion          : D l g S t a r t                                     *
**_____________________________________________________________________________**
*  Aufgabe           : Steuert die Bearbeitung einer Dialog-Box             *
*  Eingabe-Parameter: DPTR = Zeiger auf eine Dialogstruktur                *
*  Return-Wert       : Terminations-Code                                    *
***************************************************************************/
BYTE DlgStart( DIGDESPTR dptr )
{
 BYTE      i,                                             /* Schleifenzähler */
           mx,                      /* Position des Mauscursors als Koordinate */
           my;                      /* in Bezug auf den gesamten Bildschirm   */
 int       event;                             /* Ereignis von KbmEventWait() */
 void   ** lidptr;                              /* Laufzeiger in IDPTR-Vektor */
 DLGDPTR   dlgdptr;          /* Laufzeiger in Vektor auf den dp.datptr zeigt */
 TASTE     key;                                          /* empfangene Taste */

 dp = *dptr;                  /* Dialogstruktur in globale Variable kopieren */
 MouHideMouse();                                     /* Maus-Cursor ausblenden */
 VioWinOpen( dp.x_start, dp.y_start,         /* Fenster für Dialogbox öffnen */
             dp.x_start + dp.x_len - 1, dp.y_start + dp.y_len - 1 );
 VioFrame( VL(0), VO(0), VR(0), VU(0), EINRA, F(dlgbox) );
 VioFill( VL(1), VO(1), VR(-1), VU(-1), ' ', F(dlgbox) );

 if ( dp.paintfkt != NOPAINTFKT )              /* gibt es eine Paint-Funktion? */
  (*dp.paintfkt)();                              /* Ja, diese Funktion aufrufen */

 /*— Vektor zur Aufnahme der Identifikations-Pointer allokieren und Dia-  -*/
 /*— logfunktion STARTFKT der einzelnen Dialog-Felder aufrufen             -*/

 lidptr = idptr = (void **) malloc( sizeof( void * ) * dp.anz );
 for ( dlgdptr = dp.datptr, i = 0; i < dp.anz; ++dlgdptr, ++i )
  *lidptr++ = FKTL(start)( (dlgdptr)->daten );

 /*— Eingabeschleife, zunächst aktives Dialog-Feld ermitteln ————————————*/
 for ( aktiv = 0, lidptr = idptr, dlgdptr = dp.datptr;
       ! FKTL(can)( *lidptr );
       ++aktiv, ++dlgdptr, ++lidptr )
  ;
 MouShowMouse();                                 /* Maus-Cursor wieder anzeigen */

 aktip   = *lidptr;      /* Zeiger auf Datenblock des aktiven Feldes merken */
 aktdlgp = dlgdptr;  /* Zeiger auf Daten mit Infos über aktives Feld merken */
 FKT(aktiv)( aktip, TF_FELD_VOR );   /* ausgewähltes Dialog-Feld aktivieren */
 maus = ende = FALSE;
 do
  {
   if ( maus )                            /* muß die Maus überwacht werden? */
    {                        /* Ja, Ereignis von Maus oder Tastatur erwarten */
     event = KbmEventWait( EV_KEY_AVAIL | EV_MOU_ALL );
     if ( event & EV_KEY_AVAIL )                            /* Taste betätigt? */
      DlgiProcessKey( KbdGetKey() );                /* Ja, holen und auswerten */
     else                                               /* Nein, Maus-Ereignis */
      {                 /* Dialog-Feld suchen, das sich der Maus bemächtigt */
       mx = MouGetCol();                          /* Position des Maus-Cursors */
       my = MouGetRow();                          /* holen und merken          */
       retcode = FKT(maus)( aktip, mx, my, event );  /* Mausev. übermitteln */
       if ( retcode == TF_WEITER )           /* wurde das Event verarbeitet? */
        {                   /* Nein, Feld suchen, das das Ereignis akzeptiert */
         for ( i = 0, lidptr = idptr, dlgdptr = dp.datptr;
               i < dp.anz;
               ++i, ++dlgdptr, ++lidptr )
          if ( i != aktiv  &&
               FKTL(maus)( *lidptr, mx, my, event ) == TF_MAUS )
           break;
         if ( i < dp.anz )                    /* wurde ein Dialog-Feld gefunden */
          {                                                             /* Ja */
           FKT(deak)( aktip );                   /* aktuelles Feld deaktivieren */
           aktip   = *lidptr;  /* Ptr auf Datenblock des akt. Feldes merken */
           aktdlgp = dlgdptr;        /* Ptr auf Daten über akt. Feld merken */
           aktiv   = i;                            /* Nummer des Feldes merken */
           FKT(aktiv)( aktip, TF_MAUS );              /* neues Feld aktivieren */
          }
         else                    /* kein Feld nimmt die Maus in Anspruch */
          maus = FALSE;                        /* Maus nicht mehr überwachen */
        }
       else                                         /* kein TF_WEITER emfangen */
        ende = ( retcode != TF_ACCEPTED );                     /* terminieren? */
      }
    }
   else
    {       /* Nein, auf Taste oder Betätigung des linken Mausknopfs warten */
     event = KbmEventWait( EV_KEY_AVAIL | EV_LEFT_PRESS );
     if ( event & EV_KEY_AVAIL )                            /* Taste betätigt? */
      DlgiProcessKey( KbdGetKey() );                /* Ja, holen und auswerten */
     else                      /* Nein, linker Mausknopf wurde niedergedrückt */
      {                 /* Dialog-Feld suchen, das sich der Maus bemächtigt */
       mx = MouGetCol();                          /* Position des Maus-Cursors */
       my = MouGetRow();                          /* holen und merken          */
       for ( i = 0, lidptr = idptr, dlgdptr = dp.datptr;
             i < dp.anz  &&
             FKTL(maus)( *lidptr, mx, my, EV_LEFT_PRESS ) != TF_MAUS;
             ++i, ++dlgdptr, ++lidptr )
        ;
       if ( i < dp.anz )                      /* wurde ein Dialog-Feld gefunden */
        {                                                               /* Ja */
         FKT(deak)( aktip );                     /* aktuelles Feld deaktivieren */
         aktip   = *lidptr;   /* Ptr auf Datenblock des akt. Feldes merken */
         aktdlgp = dlgdptr; /*Ptr auf Daten mit Infos über akt. Feld merken*/
         aktiv   = i;                              /* Nummer des Feldes merken */
         FKT(aktiv)( aktip, TF_MAUS );                /* neues Feld aktivieren */
         maus = TRUE;                              /* Maus jetzt überwachen */
        }
      }
    }
  }
```

und gewisse Initialisierungen vorzunehmen, die für die weitere Dateneingabe innerhalb des Objekts von Bedeutung sind.

Genau wie der Dialog-Manager ein Objekt von seiner Aktivierung durch den Aufruf der AKTIVFKT in Kenntnis setzt, genau so ruft er auch die DEAKFKT auf, um einem Objekt mitzuteilen, daß es deaktiviert wurde. Für das Objekt verbindet sich mit dem Aufruf dieser Dialog-Funktion damit die Forderung, sich auf dem Bildschirm als inaktiv zu markieren, was immer das im Fall der verschiedenen Objekte auch bedeuten mag.

Der Verarbeitung von Tastatureingaben dient die TASTFKT. Sobald der Dialog-Manager – und nicht etwa ein Objekt selbst – eine Taste empfängt, reicht er diese an die TASTFKT des in diesem Moment aktiven Objekts weiter. Wie das Objekt diese Taste verarbeitet, welche Veränderungen sie innerhalb des Objekts auslöst, ist dem Dialog-Manager völlig egal. Für ihn zählt allein der Return-Code der Funktion, der ihn darüber in Kenntnis setzt, ob die Funktion die Taste verarbeiten konnte oder nicht.

Dieser Return-Code muß einer der Konstanten TF_MAUS, TF_WEITER, TF_AKTIV etc. entsprechen, die in der Include-Datei DLG.H definiert werden. Wurde die übergebene Taste verarbeitet, das eingegebene Zeichen also beispielsweise in ein gerade aktives Edit-Objekt aufgenommen, dann sollte die TASTFKT dem Dialog-Manager den Wert TF_ACCEPTED zurückliefern.

Wertet die TASTFKT die eingegebene Taste als Aufruf zur Aktivierung des nächsten oder vorhergehenden Objekts (Tab und Shift-Tab), kann sie den Wert TF_FELD_VOR bzw. TF_FELD_RUECK zurückliefern. Der Dialog-Manager sucht in diesem Fall mit Hilfe der CANFKT nach dem nächsten zu aktivierenden Objekt und ruft, sobald er fündig wird, zunächst die DEAKFKT des bisher aktiven Objekts und die AKTIVFKT des neu zu aktivierenden Objekts auf.

Konnte die Taste durch das aktive Objekt nicht verarbeitet werden, muß dem Aufrufer der Wert TF_WEITER übergeben werden. Er veranlaßt den Dialog-Manager, die Taste an alle anderen Objekte weiterzureichen, bis ein anderes Objekt auf diese Taste reagiert oder sich herausstellt, daß kein Objekt auf diese Taste eingehen möchte. Da er für diesen Test jeweils wiederum die TASTFKT der verschiedenen Objekte aufruft, muß die TASTFKT grundsätzlich in zwei Modi arbeiten. Im ersten Modus (Objekt ist aktiv) versteht das Objekt den Aufruf der TASTFKT als Aufforderung, die Taste zu verarbeiten und reagiert in der bisher beschriebenen Art und Weise.

Ist das Objekt beim Aufruf der TASTFKT allerdings nicht aktiv, ist zuvor also kein Aufruf der AKTIVFKT erfolgt, sollte es diesen Aufruf als Frage des Dialog-Managers verstehen, ob die übergebene Taste zur Aktivierung des Objekts führen soll – sie also beispielsweise den Hotkey des Objekts widerspiegelt. Trifft dies zu, muß das

◄◄ Listing 2: Fortsetzung

```
 while ( !ende );                  /* wiederholen, bis Ende-Flag gesetzt wurde */

 /*— die einzelnen Dialogfelder durchlaufen und die Dialog-Funktion ———————*/
 /*— ENDEFKT aufrufen                                                     —*/

 MouHideMouse();                                  /* Maus-Cursor ausblenden */
 lidptr = idptr;    /* jeweils den von STARTFKT retournierten Zeiger überg. */
 for ( dlgdptr = dp.datptr, i = 0;
       i < dp.anz;
       ++dlgdptr, ++i )
  ( *(dlgdptr)->fkts->ende )( *lidptr++  );

 VioWinClose( TRUE );                       /* Dialogfenster wieder schließen */
 MouShowMouse();                             /* Maus-Cursor wieder einblenden */
 return retcode;                                  /* Return-Code zurückliefern */
}


/****************************************************************************
*  Funktion         : D l g P r i n t                                       *
**__________________________________________________________________________**
*  Aufgabe          : Steht allen Arten von Dialog-Feldern zur Verfügung,   *
*                     die einen Text innerhalb der Dialog-Box ausgeben      *
*                     möchten.                                              *
*                     Trifft die Funktion innerhalb des auszugebenden Textes *
*                     auf das Zeichen HOTKEY, betrachtet es den nachfolgen- *
*                     den Buchstaben als den Hotkey des Dialog-Feldes und   *
*                     liefert dessen Tastaturcode in Verbindung mit der ALT- *
*                     Taste zurück                                          *
*  Eingabe-Parameter: x, y  = Ausgabeposition in Bezug auf den gesamten     *
*                             Bildschirm.                                   *
*                     str   = Zeiger auf den auszugebenden String.          *
*                     fn    = Farbe für normale Zeichen.                    *
*                     fh    = Farbe für den Hotkey.                         *
*  Return-Wert      : NOKEY, wenn kein Hotkey entdeckt wurde, sonst der Tas- *
*                     taturcode des Hotkeys.                                *
****************************************************************************/

TASTE DlgPrint( BYTE x, BYTE y, char * str, BYTE fn, BYTE fh )
{
 static TASTE altcodes[] =                          /* Codes der Alt-Tasten */
  {
   ALT_A, ALT_B, ALT_C, ALT_D, ALT_E, ALT_F, ALT_G, ALT_H,
   ALT_I, ALT_J, ALT_K, ALT_L, ALT_M, ALT_N, ALT_O, ALT_P,
   ALT_Q, ALT_R, ALT_S, ALT_T, ALT_U, ALT_V, ALT_W, ALT_X,
   ALT_Y, ALT_Z
  };
 TASTE retkey;                                  /* zurückgelieferter Hotkey */

 retkey = NOKEY;                              /* nicht von Hotkey ausgehen */
 while ( *str )                            /* den Ausgabestring durchlaufen */
  {
   if ( *str == HOTKEY )                              /* Hotkey entdeckt? */
    {                                                               /* Ja */
     retkey = altcodes[ tolower( *++str ) - 'a' ];         /* Hotkey merken */
     VioPrintf(x++, y, fh, FALSE, "%c", *str++ );        /* Hotkey ausgeben */
    }
   else                                                       /* kein Hotkey */
     VioPrintf( x++, y, fn, FALSE, "%c", *str++ );      /* Zeichen ausgeben */
  }
 return retkey;                                 /* den Hotkey zurückliefern */
}


/****************************************************************************
*  Funktion         : D l g D e l a y                                       *
**__________________________________________________________________________**
*  Aufgabe          : Realisiert eine Zeitverzögerung auf der Basis des BIOS *
*                     Timers.                                               *
*  Eingabe-Parameter: PAUSLEN = Länge der Pause in Ticks                    *
*                               (1 Ticks entspricht 1/18,2 sec.)            *
*  Return-Wert      : keiner                                                *
****************************************************************************/

void DlgDelay( int pauslen )
{
 register unsigned int zeit_hi,                                /* Zeitzähler */
                       zeit_lo;
 union REGS            inregs,                          /* Prozessorregister */
                       outregs;

 inregs.h.ah = 0;                      /* Funktion 00h = Zeitzähler auslesen */
 int86( 0x1a, &inregs, &outregs );                  /* Zeit holen und merken */
 zeit_hi = outregs.x.cx;
 zeit_lo = outregs.x.dx;

 while ( pauslen )                        /* wiederholen, bis pauslen auf 0 */
  {                                       /* heruntergezählt wurde          */
   int86( 0x1a, &inregs, &outregs );                          /* Zeit holen */

   /*— neuer Tick angebrochen?——————————————————————————————————————— */

   if ( zeit_hi != outregs.x.cx  ||  zeit_lo != outregs.x.dx )
    {                                                               /* Ja */
     zeit_hi = outregs.x.cx;              /* neue Werte der Zeitzähler merken */
     zeit_lo = outregs.x.dx;
     --pauslen;                   /* Anzahl verbleibender Ticks dekrementieren */
    }
  }
}


/****************************************************************************
*  Funktion         : D l g B o x                                           *
**__________________________________________________________________________**
*  Aufgabe          : Zieht einen Rahmen innerhalb einer Dialogbox und setzt *
*                     einen Titel hinein.                                   *
*  Eingabe-Parameter: X, Y  = Start des Rahmens relativ zu oberen linken    *
*                             Ecke der Dialogbox                            *
*                     XLEN, = Breite des Rahmens                            *
*                     YLEN, = Höhe des Rahmens                              *
*                     TYP   = Rahmentyp (EINRA, DOPRA etc.)                 *
*                     TITEL = der Titel, der in der ersten Rahmenzeile aus- *
*                             gegeben wird                                  *
*                     AKTIV = TRUE, Rahmen und Titel werden in den Farben   *
*                             aktiver Felder gezeichnet.                    *
*  Return-Wert      : keiner                                                *
*  Info             : Hotkey-Markierungen im Titel werden nicht berücksich- *
*                     tigt und wie normale Zeichen ausgegeben.              *
****************************************************************************/

void DlgBox( BYTE x, BYTE y, BYTE xlen, BYTE ylen,
             BYTE typ, char * titel, BOOL aktiv )

{
 BYTE f;                                                   /* Ausgabefarbe */

 VioFrame( VL(x), VO(y), VL(x+xlen-1), VO(y+ylen-1),
           typ, f = aktiv ? F(hi) : F(nm) );
 VioPrint( VL( x+(xlen-strlen(titel) >> 1) ), VO( y ), f, FALSE, titel );
}
```

►► *Listing 3: DLGEDIT.C*

Objekt an den Dialog-Manager den Wert TF_AKTIV zurückliefern, um ihn zu seiner Aktivierung aufzufordern. Andernfalls wird auch hier TF_WEITER übergeben, damit der Dialog-Manager die Taste an andere Objekte zur Überprüfung weiterreicht.

Werden Tastaturereignisse mit Hilfe der TASTFKT bearbeitet, so stellt jedes Objekt für die Behandlung von Mausereignissen die MAUSFKT zur Verfügung, die ganz wie die TASTFKT in zwei Modi arbeitet.

Soweit ein Objekt noch nicht aktiv ist, genauer gesagt, nicht über die Maus aktiviert wurde, muß es den Aufruf dieser Funktion als Frage interpretieren, ob es sich durch das übergebene Mausereignis angesprochen fühlt, d.h. aktiviert werden möchte.

Um diese Frage beantworten zu können, übergibt der Dialog-Manager der MAUSFKT neben der aktuellen Position des Maus-Cursors auch die Event-Maske, die die Funktion KbmEventWait() (Abfrage von Maus und Tastatur) zurückgeliefert hat. Erkennt sich das Objekt als Adressat des Mausereignisses, muß es TF_MAUS zurückliefern, um aktiviert zu werden.

Anderenfalls verlangt auch hier der Dialog-Manager die Rückgabe von TF_WEITER, damit er das Ereignis an ein anderes Objekt weiterreichen kann.

Ist das Objekt beim Aufruf der MAUSFKT hingegen bereits aktiv, so muß es auf das übergebene Ereignis eingehen und – sofern es dieses Ereignis auf sich beziehen kann – TF_ACCEPTED zurückliefern. Anderenfalls veranlaßt die Rückgabe von TF_WEITER auch hier den Dialog-Manager, sein Glück bei den anderen Objekten der Dialogbox zu versuchen.

Liefern MAUSFKT oder TASTFKT einen anderen Wert als TF_WEITER, TF_ACCEPTED etc. zurück, versteht dies der Dialog-Manager als Aufforderung zum Schließen die Dialogbox und liefert den übergebenen Code als Return-Wert an den Aufrufer von DlgStart() zurück. Wie wir im folgenden noch sehen werden, machen von dieser Möglichkeit jedoch nur die Action-Buttons Gebrauch, die allein in der Lage sind, die Dateneingabe innerhalb einer Dialogbox zu beenden.

Eine letzte Dialog-Funktion, die NEWVALFKT, ist bisher nicht zu Wort gekommen, weil sie vom Dialog-Manager selbst gar nicht aufgerufen wird. In Standard-Objekten, die miteinander nicht in Verbindung stehen, wird sie nicht benötigt und wird deshalb durch einen Dummy repräsentiert. Im Rahmen von Dialogboxen mit interdependenten Feldern kann sie jedoch von den Funktionen des Programmierers herangezogen werden, um dem Objekt mitzuteilen, daß sich sein Inhalt durch äußere Ereignisse verändert hat. Beispiele dafür werde ich den kommenden Folgen dieser Serie bei der weiteren Beschreibung der Dialogobjekte geben.

```
/***********************************************************************/
/*                          D L G E D I T . C                          */
/*---------------------------------------------------------------------*/
/*    Aufgabe        : Enthält die Funktionen zur Verwaltung von EDIT-   */
/*                     Feldern, die innerhalb von Dialog-Boxen zum Einsatz */
/*                     kommen.                                          */
/*                     Dieses Modul muß in Verbindumg mt dem DLG-Modul zum */
/*                     Einsatz kommen.                                  */
/*---------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                  */
/*    entwickelt am  : 13.07.1989                                       */
/*    letztes Update : 17.07.1989                                       */
/*---------------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] DLGEDIT.C /C                    */
/*                     dann mit dem DLG-Modul und anderen Moduln verbinden */
/***********************************************************************/

/*— Include-Dateien einbinden ——————————————————————————————————————————*/

#include <string.h>
#include <memory.h>
#include <malloc.h>
#include <dos.h>
#include "dlg.h"

/*— interne Datenstrukturen des Dialogtyps EDIT ————————————————————————*/

typedef struct  /* diese Daten werden für jedes einzelne EDIT-Feld festgeh. */
 {
  BOOL     aktiv,                        /* ist das Dialog-Feld gerade aktiv? */
           maus,                         /* linker Mausknopf niedergedrückt? */
           insert;                               /* zeigt INSERT-Modus an */
  BYTE     x,        /* Nr der Bildschirmspalte am linken Rand des E-Feldes */
           y,                                              /* Ausgabezeile */
           xlinks,           /* Nummer der ersten sichtbaren Eingabespalte */
           xcur;     /* Cursorpos. relativ zur linken Rand des Eingabefelds */
  TASTE    hotkey;           /* ist 0, wenn das Feld keinen Hotkey besitzt */
  EDITFELD d;                            /* die Daten aus der Edit-Struktur */
 } EDITINTERN;

typedef EDITINTERN * EDITP;         /* Zeiger auf eine interne EDIT-Struktur */

/*— Konstanten —————————————————————————————————————————————————————————*/

#define FILL_CHAR 255         /* Füll-Zeichen zur Unterscheidung von Spaces */
#define INSERT    338                   /* INSERT-Taste, in KBM.H vergessen */

/*— Prototypen der EDIT-Funktionen, die in diesem Modul deklariert und in -*/
/*— der globalen Variable std_edit_fkt zusammengefaßt werden             -*/

void * edit_start ( EDITFELD * dptr );
void   edit_newval( EDITP ep, void * dptr );
void   edit_ende  ( EDITP ep );
BYTE   edit_taste ( EDITP ep, TASTE key );
void   edit_deak  ( EDITP ep );
void   edit_aktiv ( EDITP ep, BYTE why );
BYTE   edit_maus  ( EDITP ep, BYTE x, BYTE y, BYTE ev );
BOOL   edit_can   ( EDITP ep );

/*— globale Variablen, öffentlich ——————————————————————————————————————*/

DLGFKT std_edit_fkt =                     /* Standard Dialog-Funktionen für */
 {                                        /* ein EDIT-Dialogfeld            */
   edit_start, edit_newval, edit_ende , edit_taste,
   edit_deak,  edit_maus,   edit_aktiv, edit_can
 };


/***********************************************************************
* E D I T       Es folgen die verschiedenen Funktionen des Dialogtyps EDIT *
*               ( alphanumerische Eingabefelder )                        *
***********************************************************************/


/***********************************************************************
*  Funktion         : E d S e t M e n g e                              *
**---------------------------------------------------------------------**
*  Aufgabe          : Dient einem Programm, das eine Dialog-Box erstellen *
*                     möchte, bei der Definition der Eingabemenge für ein *
*                     Edit-Feld                                        *
*  Eingabe-Parameter: MPTR = Zeiger auf den Puffer der Eingabemenge     *
*                     VON  = ASCII-Code des ersten erlaubten Zeichens   *
*                     BIS  = ASCII-Code des letzten erlaubten Zeichens  *
*                     SET  = TRUE wenn die Zeichen erlaubt sein sollen, *
*                            sonst FALSE                                *
*  Return-Wert      : keiner                                           *
*  Info             : Die Funktion darf beliebig oft aufgerufen werden, um *
*                     nacheinander verschiedene Zeichenbereiche als erlaubt *
*                     zu markieren.                                    *
***********************************************************************/

void EdSetMenge( EMP mptr, BYTE von, BYTE bis, BOOL set )
{
 int  imin,               /* Indexadressen für Zugriff auf den Mengen-Vektor */
      imax;
 BYTE i;                                               /* Schleifenzähler */

 imin = von + 7 >> 3;                /* Inidizes für Zugriffe auf den Mengen- */
 imax = bis - 7 >> 3;                /* Vektor über MEMSET berechnen          */
 if ( imax > imin )               /* wird mindestens ein Byte komplett belegt? */
  {                                        /* Ja,  Bytes imin-imax setzen */
   memset( mptr + imin, set ? 0xff : 0, imax - imin + 1 );

   /*— die Bits setzen, die vor IMIN liegen———————————————————————— */
   if ( ( von & 7 ) != 0)                          /* gibt es Bits vor IMIN? */
    {                                                               /* ja */
     if ( set )                                            /* Bits setzen? */
      *(mptr + imin - 1) |= ~( 0xff << ( 8 - ( von & 7 ) ) );         /* Ja */
     else                                          /* Nein, Bits löschen */
      *(mptr + imin - 1) &= ( 0xff << ( 8 - ( von & 7 ) ) );
    }

   /*— die Bits setzen, die hinter IMAX liegen————————————————————— */

   if ( ( bis & 7 ) != 7)                         /* gibt es Bits hinter IMAX? */
    {                                                               /* Ja */
     if ( set )                                             /* Bits setzen */
      *(mptr + imax + 1) |= ( 0xff << ( 7 - ( bis & 7 ) ) );          /* Ja */
     else                                               /* Nein, löschen */
      *(mptr + imax + 1) &= ~( 0xff << ( 7 - ( bis & 7 ) ) );
    }
  }
 else                                        /* nein die Bits einzeln setzen */
  if ( set )                                   /* Bits setzen oder löschen? */
   for ( i = von ; i <= bis ; ++i )                                /* setzen */
    *(mptr + ( i >> 3 )) |= ( 1 << ( 7 - ( i & 7 ) ) );
  else                                                            /* löschen */
   for ( i = von ; i <= bis ; ++i )
    *(mptr + ( i >> 3 )) &= ~( 1 << ( 7 - ( i & 7 ) ) );
}
```

◂◂ *Listing 3: Fortsetzung*

```
/***************************************************************************
*  Funktion           : e d i t _ f i l l                                  *
**_______________________________________________________________________**
*  Aufgabe            : Füllt den Eingabe-String eines Edit-Feldes auf seine  *
*                       maximale Länge mit dem Füllzeichen FILL_CHAR auf.     *
*  Eingabe-Parameter: EP  = Zeiger auf eine interne EDIT-Datenstruktur.      *
*  Return-Wert        : keiner                                             *
*  Info               : Diese Funktion findet nur intern Verwendung, sie wird *
*                       nicht vom Scheduler aufgerufen.                    *
***************************************************************************/

void edit_fill( EDITP ep )
{
 BYTE  lstr,                                  /* Länge des Eingabe-Strings */
       len,                          /* maximale Länge des Eingabe-Strings */
     * sp;                                   /* Zeiger auf den Eingabstring */

 /*— Eingabe-String mit gesperrten Leerzeichen (ASCII 255 ) auffüllen ——*/
 lstr = strlen( sp = ep->d.eingabe );   /* Länge des Eingabe-Str. ermitteln */
 memset( sp + lstr, FILL_CHAR, ( len = ep->d.len ) - lstr );   /* auffüllen */
 *( sp + len ) = '\0';                            /* Ende-Markierung setzen */
}

/***************************************************************************
*  Funktion           : e d i t _ s h r i n k                              *
**_______________________________________________________________________**
*  Aufgabe            : Entfernt die bei edit_fill eingefügten Leerzeichen   *
*                       aus einem Edit-String wieder und schneidet den String *
*                       nach dem letzten eingegebenen nicht-Füllzeichen ab.   *
*  Eingabe-Parameter: EP  = Zeiger auf eine interne EDIT-Datenstruktur.      *
*  Return-Wert        : keiner                                             *
*  Info               : Diese Funktion findet nur intern Verwendung, sie wird *
*                       nicht vom Scheduler aufgerufen.                    *
***************************************************************************/

void edit_shrink( EDITP ep )
{
 BYTE   i,                                          /* Schleifenzähler */
      * lp,                           /* Laufzeiger in den Eingabestring */
      * sp;                    /* Zeiger auf den Anfang des Eingabestrings */

 lp = ( sp = ep->d.eingabe ) + ( i = ep->d.len ) - 1;    /* von hinten nach */
 for (  ; i  && *lp == FILL_CHAR; —i, —lp )              /* vorne suchen    */
  ;
 *( lp+1 ) = '\0';                                    /* String abschließen */

 /*— alle vorhergenden FILL_CHARs in Spaces umwandeln ——————————*/
 for ( ; lp > sp ; —lp )          /* den String bis zum Anfang durchlaufen */
  if ( *lp == FILL_CHAR )                            /* FILL_CHAR entdeckt? */
   *lp = ' ';                                      /* Ja, in Space umwandeln */
}

/***************************************************************************
*  Funktion           : e d i t _ t o g g l e _ i n s e r t                *
**_______________________________________________________________________**
*  Aufgabe            : Schaltet den Insert-Status eines EDIT-Feldes um und   *
*                       definiert einen Block-Cursor, falls der INSERT-Modus  *
*                       aktiviert wird.                                    *
*  Eingabe-Parameter: EP  = Zeiger auf eine interne EDIT-Datenstruktur.      *
*  Return-Wert        : keiner                                             *
*  Info               : Diese Funktion findet nur intern Verwendung, sie wird *
*                       nicht vom Scheduler aufgerufen.                    *
***************************************************************************/

void edit_toggle_insert( EDITP ep )
{
 union REGS   regs;                  /* Prozessorregister zum Interruptaufruf */

 /*— Register CL und CH mit Start- und Endzeile des Cursors laden ————*/
 regs.h.cl = VioIsColor() ? 7 : 13;                      /* Cursor-Endzeile */
 regs.h.ch = (ep->insert = !ep->insert) ? 0 : regs.h.cl-1;    /* Startzeile */
 regs.h.ah = 0x01;                                     /* Cursor definieren */
 int86( 0x10, &regs, &regs );                         /* Video-BIOS aufrufen */
}

/***************************************************************************
*  Funktion           : e d i t _ p r i n t                                *
**_______________________________________________________________________**
*  Aufgabe            : Gibt den EDIT-String eines EDIT-Feldes auf dem Bild-  *
*                       schirm aus.                                        *
*  Eingabe-Parameter: EP    = Zeiger auf eine interne EDIT-Datenstruktur.    *
*                       FARBE = Ausgabefarbe des Strings                   *
*  Return-Wert        : keiner                                             *
*  Info               : - Diese Funktion findet nur intern Verwendung, sie   *
*                         wird nicht vom Scheduler aufgerufen.             *
*                       - Ausgabeposition und auszugebender Teil des EDIT-   *
*                         Strings ergeben sich aus den Informationen in der  *
*                         übergebenen Datenstruktur.                       *
***************************************************************************/

void edit_print( register EDITP ep, BYTE farbe )
{
 BYTE   merke,                      /* merkt sich ein Zeichen aus dem String */
      * mpos,                             /* Zeiger auf das Merke-Zeichen */
      * start;                                        /* Ausgabeposition */

 /*— Zeichen mit Hilfe von VioPrint ausgeben, indem hinter das letzte ———*/
 /*— auszugebende Zeichen eine Ende-Markierung geschrieben wird       ———*/

 merke = *(mpos = (start=ep->d.eingabe+ep->xlinks) + ep->d.visi );
 *mpos = '\0';                           /* Ende-Markierung für VioPrint setzen */
 VioPrint( ep->x, ep->y, farbe, FALSE, start );           /* String ausgeben */
 *mpos = merke;                                /* altes Zeichen zurückschreiben */
}

/***************************************************************************
*  Funktion           : e d i t _ s t a r t                                *
**_______________________________________________________________________**
*  Aufgabe            : Wird vom Scheduler während der Initialisierung einer  *
*                       Dialogbox für jedes EDIT-Feld aufgerufen.          *
*  Eingabe-Parameter: DPTR = Zeiger auf den Datenblock des EDIT-Feldes.      *
*  Return-Wert        : Ein Zeiger, den der Scheduler bei allen folgenden Auf- *
*                       rufen den Funktionen edit_aktiv, edit_tast etc. über- *
*                       gibt.                                              *
***************************************************************************/

void * edit_start ( EDITFELD * dptr )
{
 EDITP ep;                   /* Zeiger auf die anzulegende interne Datenstruktur */
 BYTE  farbe_n,                                     /* Ausgabefarbe für Feld */
       farbe_h,                       /* Farbe des Hotkeys (soweit vorhanden) */
       s,                             /* nimmt Startspalte des Eingabefeldes auf */
       z;                                          /* nimmt Ausgabezeile auf */

 /*— Speicherplatz für interne Struktur allok. und Struktur initialisieren */

 ep = (EDITP) malloc( sizeof( EDITINTERN ) );   /* Speicherplatz allokieren */
 z = ep->y = VO( dptr->y );                           /* Ausgabezeile merken */
 ep->xlinks = ep->xcur = 0; /* Cursor am linken Rand des Feldes in Spalte 0 */
 ep->maus = ep->aktiv = ep->insert = FALSE;                /* alle Flags FALSE */

 ep->d = *dptr;                                      /* EDIT-Struktur kopieren */
```

Farben

Mehrfach wurde bereits erwähnt, daß aktive Felder gehalten sind, sich innerhalb der Dialog-Box farblich hervorzuheben. Die dabei verwendeten Farben, müssen sie aus einer Struktur mit dem Namen DLGCOL beziehen, die innerhalb der Include-Datei DLG.H definiert wird. Hier finden sich neben der Farbe, mit der der Dialog-Manager die Dialogbox füllt und den Rahmen um die Dialogbox zieht, noch die fünf folgenden Farben:

nm	Farbe für inaktive Dialogfelder und Texte
hi	Farbe für hervorgehobene Dialogfelder
hk	Farbe für den Hotkey innerhalb des Texts, der zu einem Objekt gehört
da	Farbe für Objekte, die nicht aktiviert werden können
dk	Farbe für den Hotkey in Objekten, die nicht aktiviert werden können

Innerhalb des DLG-Moduls wird eine globale Variable vom Typ DLGCOL definiert, die den Namen dlgcol trägt und auf die sich nicht nur der Dialog-Manager, sondern auch die verschiedenen Objekte bei Ausgaben innerhalb der Dialogbox beziehen müssen. Bedienen können Sie sich dazu des Makros F(), das innerhalb der Include-Datei DLG.H definiert wird. Es ermöglicht den Zugriff auf die Komponente innerhalb der dlgcol-Variable, die dem Makro in Klammern übergeben wird. F(hi) liefert so den Inhalt von dlgcol.hi.

Für den Monochrom-Modus ist dlgcol übrigens bereits initialisiert, aber wenn Sie sich anderer Farben bedienen möchten oder ein Color-Adapter angeschlossen ist, können Sie natürlich jederzeit andere Farbwerte in die einzelnen Komponenten dieser Variable eintragen.

Kleine Helferlein

Neben DlgStart() beinhaltet das DLG-Modul drei weitere Funktionen, die die verschiedenen Dialog-Funktionen bei ihrer Arbeit unterstützen. Es sind dies die Funktionen DlgPrint(), DlgDelay() und DlgBox().

DlgPrint() hilft bei der Ausgabe des zu einem Objekt gehörenden Texts, indem es diesen Text nicht nur ausgibt, sondern gleichzeitig auch den Hotkey sucht, der durch ein vorangehendes »#«-Zeichen markiert wird. Den Tastaturcode dieses Hotkeys in Verbindung mit der Alt-Taste liefert es an den Aufrufer zurück.

DlgDelay() hält die Programmausführung für einen bestimmten Zeitraum an, was z.B. nach der Reaktion auf ein Mausereignis nützlich sein kann, damit sich dieses Ereignis nicht bereits im nächsten Augenblick wiederholt (etwa weil der Anwender den linken Mausknopf weiterhin niedergedrückt hält). Die Funktion bedient sich dazu der Funktion 00h des BIOS-Timer-Inter-

▸▸ Listing 3: Fortsetzung

rupts 1Ah. Sie liefert die aktuelle Uhrzeit in Form eines Zählers zurück, der nach dem Systemstart 18,2 mal in der Sekunde inkrementiert wird, bis er um Mitternacht wieder auf 0 gesetzt wird.

DlgBox() schließlich hilft bei der Gestaltung der Dialogbox, wenn es darum geht, einen Rahmen um eine Gruppe von Objekten zu ziehen und diesen Rahmen mit einem Titel zu versehen.

Realisation der Objekte

Neben dem Dialog-Manager finden Sie in dieser Folge bereits die Realisation des Edit-Objekts in der Datei DLGEDIT.C und der Action-Button in der Datei DLGAB.C.

Beide Dateien enthalten nicht viel mehr als die vom Dialog-Manager benötigten Dialog-Funktionen und einige zusätzliche Hilfsfunktionen, die jedoch nur intern Verwendung finden. Für beide Arten von Objekten findet sich in DLG.H eine Struktur, die die benötigten Informationen für die Verwaltung der Objekte aufnimmt. Für Edit trägt diese Struktur den Namen EDITFELD und für die Action-Buttons heißt sie ABGROUP. Auf die Bedeutung der einzelnen Felder für die Definition des Objekts muß an dieser Stelle nicht weiter eingegangen werden – sie geht aus den Kommentaren in DLG.H eindeutig hervor.

Einzig und allein der Typ EMENGE, den das Edit-Objekt zur Bestimmung der bei der Eingabe erlaubten Zeichen benötigt, bedarf hier noch einer Klärung. EMENGE wird als ein Vektor bestehend aus 64 Bytes definiert und stellt aus der Sicht des Edit-Objekts ein Bit-Feld mit 256 Einträgen (64 Byte * 8 Bits = 256 Bits) dar. Jeder dieser Einträge korrespondiert mit dem Zeichen aus dem 256 Zeichen umfassenden ASCII-Zeichensatz des PC, das den Code der Bitnummer trägt. Ist das entsprechende Bit gesetzt, darf das Zeichen eingegeben werden, sonst nicht.

Um die Definition einer solchen Eingabemenge zu erleichtern, hält das Modul DLGEDIT die Funktion EdSetMenge() bereit. Ihr muß neben einem Zeiger auf die Eingabe-Menge nur die Nummer des ersten und letzten zu bearbeitenden ASCII-Codes sowie ein Flag übergeben werden, daß anzeigt, ob die adressierten Zeichen zur Eingabe erlaubt sein sollen oder nicht. Innerhalb der Eingabemenge setzt sie dann die entsprechenden Bits bzw. löscht sie. Und um die Arbeit mit dieser Funktion noch ein wenig komfortabler zu gestalten, enthält DLG.H verschiedene Makros (SetMengeInt, SetMengeAN, SetMengeDatei etc.) die die verschiedenen Aufrufe der EdSetMenge()-Funktion erzeugen, derer es zur Definition einer bestimmten Eingabemenge bedarf.

Eine kurze Anmerkung auch noch zu den Action-Buttons: wie oben bereits erwähnt, können nur Action-Buttons zum Schließen der Dialogbox führen. Damit der Aufrufer von DlgStart() erkennen kann, welcher Action-Button aufgeru

```
/*— Eingabefeld auf dem Bildschirm aufbauen ———————————————————————*/
 farbe_n = ( dptr->enabled ) ? F(nm) : F(da);     /* Ausgabefarbe festlegen */
 farbe_h = ( dptr->enabled ) ? F(hk) : F(dk);     /* Ausgabefarbe festlegen */

 ep->hotkey = DlgPrint( VL( dptr->x ), z, dptr->text, farbe_n, farbe_h );
 s = VL( strlen( dptr->text ) + dptr->x + 1 );       /* Ausgabesp. im E-Feld */
 if ( ep->hotkey != NOKEY )                      /* wurde ein Hotkey entdeckt? */
  —s;       /* Ja, Spalte  muß dekermentiert werden, weil '#' nicht zählt */
 VioPrint( (ep->x=s)-1, z, farbe_n, FALSE, "[" );     /* Begrenzer ausgeben */
 VioPrint( s + dptr->visi , z, farbe_n, FALSE, "]" );

 edit_fill( ep );                                  /* Eingabestring auffüllen */
 edit_print( ep, farbe_n );                /* Inhalt des Eingabefelds ausgeben */
 edit_shrink( ep );                             /* String in alte Form bringen */

 return ep;                  /* Zeiger auf internen Datenblock zurückliefern */
}

/***************************************************************************
*  Funktion         : e d i t _ n e w v a l                                 *
**_____________________________________________________________________**
*  Aufgabe          : Wird nicht direkt vom Scheduler, sondern von einer mit *
*                     ihm zusammenarbeitenden Interaktions-Funktion aufge-   *
*                     rufen, wenn sich der Inhalt eines EDIT-Feldes durch    *
*                     äußeres Ereignis verändert hat.                        *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-    *
*                          rückgeliefert wurde.                              *
*  Return-Wert      : keiner                                                 *
***************************************************************************/

void edit_newval( EDITP ep, void * dptr )
{
}

/***************************************************************************
*  Funktion         : e d i t _ l i n k s                                   *
**_____________________________________________________________________**
*  Aufgabe          : Bewegt den Cursor im Edit-Feld um eine Spalte nach     *
*                     links.                                                 *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-    *
*                          rückgeliefert wurde.                              *
*  Return-Wert      : keiner                                                 *
*  Info             : - Diese Funktion findet nur intern Verwendung, sie     *
*                       wird nicht vom Scheduler aufgerufen.                 *
***************************************************************************/

void edit_links( EDITP ep )
{
 if ( ep->xcur )                          /* Cursor am linken Rand des E-Felds? */
  VioSetCursor( ep->x + —ep->xcur, ep->y );         /* Nein, nur neue Pos. */
 else                                                                /* Ja */
  if ( ep->xlinks )                        /* bereits erstes Zeichen sichtbar? */
   {                                   /* Nein, Eingabefeld nach rechts scrollen */
    —ep->xlinks;
    edit_print( ep, F(hi) );                                /* Feld neu ausgeben */
   }
}

/***************************************************************************
*  Funktion         : e d i t _ r e c h t s                                 *
**_____________________________________________________________________**
*  Aufgabe          : Bewegt den Cursor im Edit-Feld um eine Spalte nach     *
*                     rechts.                                                *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-    *
*                          rückgeliefert wurde.                              *
*  Return-Wert      : keiner                                                 *
*  Info             : - Diese Funktion findet nur intern Verwendung, sie     *
*                       wird nicht vom Scheduler aufgerufen.                 *
***************************************************************************/

void edit_rechts( EDITP ep )
{
 if ( ep->xcur != ep->d.visi-1)                        /* Cursor am r. Rand? */
  VioSetCursor( ep->x + ++ep->xcur, ep->y );         /* Nein, nur neue Pos. */
 else                              /* Ja, bereits letztes Zeichen sichtbar? */
  if ( ep->xlinks < ep->d.len - ep->d.visi )
   {                                   /* Nein, Eingabefeld nach links scrollen */
    ++ep->xlinks;
    edit_print( ep, F(hi) );                                /* Feld neu ausgeben */
   }
}

/***************************************************************************
*  Funktion         : e d i t _ e n d e                                     *
**_____________________________________________________________________**
*  Aufgabe          : Wird vom Scheduler während des Schließens der Dialog-  *
*                     box aufgerufen, um EDIT Gelgegenheit zu geben, Clean-  *
*                     Up-Arbeiten durchzuführen.                             *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-    *
*                          rückgeliefert wurde.                              *
*  Return-Wert      : keiner                                                 *
***************************************************************************/

void edit_ende( EDITP ep )
{
 free( ep );                 /* den allokierten Datenblock wieder freigeben */
}

/***************************************************************************
*  Funktion         : e d i t _ t a s t e                                   *
**_____________________________________________________________________**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um dem Dialog-Feld eine *
*                     Taste zu übergeben.                                    *
*  Eingabe-Parameter: EP    = der Zeiger, der beim Aufruf von edit_start zu- *
*                             rückgeliefert wurde.                           *
*                     TASTE = Code der zu bearbeitenden Taste                *
*  Return-Wert      : Reaktionscode (TF_...)                                 *
***************************************************************************/

BYTE edit_taste( EDITP ep, TASTE key )
{
 BYTE   i,                                                /* Schleifenzähler */
        zeichen,          /* Zeichen an letzter Eingabepos. im INSERT-Modus */
      * lp,                    /* Zeiger zum Durchlaufen des Eingabestrings */
      * sp,                    /* Zeiger auf den Anfang des Eingabe-Strings */
        retcode;                                              /* Return-Code */
 if ( ep->aktiv )                                /* ist das Dialog-Feld aktiv? */
  {                                                                     /* Ja */
   MouHideMouse();                                   /* Maus-Cursor ausblenden */
   switch ( key )                                          /* Taste untersuchen */
    {
     case TAB     :       /*———————————————————————————————————————————— Tab */
      retcode = TF_FELD_VOR;
      break;
     case BACKTAB :       /*————————————————————————————————————————— BackTab */
      retcode = TF_FELD_RUECK;
      break;
     case INSERT  :       /*——————————————————————————————— Insert umschalten */
      edit_toggle_insert( ep );                  /* Insert-Status umschalten */
      retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
      break;
```

```
case CLEFT   :     /*------------------------------------ Cursor links */
 edit_links( ep );
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

case CRIGHT  :     /*----------------------------------- Cursor rechts */
 edit_rechts( ep );
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

case CHOME   :     /*------------------------------------- Cursor Home */
 ep->xlinks = ep->xcur = 0;
 VioSetCursor( ep->x + ep->xcur, ep->y );  /* Cursor neu positionieren */
 edit_print( ep, F(hi) );                         /* Feld neu ausgeben */
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

case CEND    :     /*-------------------------------------- Cursor End */
 /*-- letztes nicht FILL_CHAR-Zeichen suchen-------------------------- */

 lp = ep->d.eingabe + ( i = ep->d.len ) - 1;         /* von hinten nach */
 for (  ; i  &&  *lp == FILL_CHAR; --i, --lp )       /* vorne suchen    */
  ;
 if ( i < ep->d.visi )   /* ist Zeichen unter den ersten VISI-Zeichen? */
  {                          /* Ja, Eingabe ab erstem Zeilen darstellen */
  ep->xlinks = 0;
  ep->xcur   = i;
  }
 else               /* Nein, Eingabe so darstellen, daß sich der Cursor */
  {                 /* in der letzten Spalte des Eingabefeldes befindet */
  ep->xlinks = i - ep->d.visi + 1;
  if ( i == ep->d.len )               /* ist die letzte Pos. beschrieben? */
   --ep->xlinks;                /* Ja, Cursor steht auf letztem Zeichen */
  ep->xcur = ep->d.visi - 1;     /* Cursor in letzte Ausgabesp. setzen */
  }
 edit_print( ep, F(hi) );                         /* Feld neu ausgeben */
 VioSetCursor( ep->x + ep->xcur, ep->y );  /* Cursor neu positionieren */
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

case DELETE  :     /*------------------------------------------ Delete */
 lp = ep->d.eingabe + ( i = ep->xlinks + ep->xcur );
 memmove( lp, lp+1, ep->d.len - i - 1);          /* Zeichen heranziehen */
 *( ep->d.eingabe + ep->d.len - 1 ) = FILL_CHAR;            /*auffüllen*/
 edit_print( ep, F(hi) );                         /* Feld neu ausgeben */
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

case BS      :     /*--------------------------------------- Backspace */
 if ( ep->xcur  ||  ep->xlinks )           /* Cursor auf erstem Zeichen? */
  {  /* Nein, Zeichen ab Cursorpos. um ein Zeichen nach links schieben */
  lp = ep->d.eingabe + ( i = ep->xlinks + ep->xcur );
  memmove( lp-1, lp, ep->d.len - i);             /* Zeichen heranziehen */
  *( ep->d.eingabe + ep->d.len - 1 ) = FILL_CHAR;           /*auffüllen*/
  if ( ep->xcur )                   /* Cursor am linken Rand des E-Felds? */
   {                                                          /* Nein */
   --ep->xcur;                          /* eine Spalte nach links schieben */
   VioSetCursor( ep->x + ep->xcur, ep->y );          /* Cursor neu pos. */
   }
  else                          /* Ja, Eingabefeld nach rechts scrollen */
   --ep->xlinks;
  edit_print( ep, F(hi) );                        /* Feld neu ausgeben */
  }
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

case CTRL_HOME:    /*------------------------------------- Ctrl + Home */
 memset( ep->d.eingabe, FILL_CHAR, ep->d.len );  /* Eingabe-Str leeren */
 edit_print( ep, F(hi) );                         /* Feld neu ausgeben */
 retcode = TF_ACCEPTED;                   /* die Taste wurde akzeptiert */
 break;

default :          /*------------------------------ jedes andere Zeichen */
 if ( key < 256 &&
      (*(ep->d.mptr + (key >> 3)) & (1 << (7 - (key & 7)))) )
  {          /* das Zeichen darf laut Eingabemenge eingegeben werden */
   i = ep->xlinks + ep->xcur;    /* Offsetposition in Eingbe-Str. ber. */
   lp = ep->d.eingabe + i;    /* Zeiger auf akt. Pos. eim Eingabe-Str. */
   if ( ep->insert )                                 /* Im Insert-Modus? */
    {                         /* Ja, letztes Zeichen muß noch leer sein */
     if ( (zeichen = *(ep->d.eingabe+ep->d.len-1)) == ' '  ||
          zeichen == FILL_CHAR )
      {                                 /* letztes Zeichen ist noch leer */
       if ( i != ep->d.len - 1 )          /* Cursor auf letztem Zeichen? */
        memmove( lp+1, lp, ep->d.len - i - 1); /*N, Zeichen wegschieben*/
       *lp = key;                       /* Taste im Eingabe-Str. speichern */
       if ( ep->xcur != ep->d.visi-1)                /* Cursor am r. Rand? */
         VioSetCursor( ep->x + ++ep->xcur, ep->y );                /* Nein */
       else                                                          /* Ja */
        if ( i != ep->d.len - 1 )         /* Cursor auf letztem Zeichen? */
          ++ep->xlinks;     /* ganzes Eingabefeld nach rechts scrollen */
       edit_print( ep, F(hi) );                       /* Feld neu ausgeben */
      }
    }
   else                                            /* nicht im Insert-Modus */
    {
     *lp = key;                       /* Taste im Eingabe-Str. speichern */
     if ( ep->xcur != ep->d.visi-1)                  /* Cursor am r. Rand? */
      {                          /* Nein, nur das neue Zeichen ausgeben */
       VioPrintf( ep->x+ep->xcur, ep->y, F(hi), FALSE, "%c", (char) key);
       VioSetCursor( ep->x + ++ep->xcur, ep->y ); /*Nein, nur neue Pos.*/
      }
     else                        /* Ja, bereits letztes Zeichen sichtbar? */
      {
       if ( i != ep->d.len - 1 )
        ++ep->xlinks;                         /* Nein, nach rechts scrollen */
       edit_print( ep, F(hi) );                       /* Feld neu ausgeben */
      }
    }
   retcode = TF_ACCEPTED;                 /* die Taste wurde akzeptiert */
  }
 else              /* die Taste wurde nicht akzeptiert, weiterreichen */
   retcode = TF_WEITER;
 break;
}
MouShowMouse();                              /* Maus-Cursor wieder anzeigen */
return retcode;                                /* Return-Code zurückliefern */
}
else                         /* Feld ist nicht aktiv, auf Hotkey testen */
 return (ep->hotkey == key  &&  ep->d.enabled) ? TF_AKTIV : TF_WEITER;
}


/***************************************************************************
* Funktion          : e d i t _ d e a k                                    *
**-----------------------------------------------------------------------**
* Aufgabe           : Wird vom Scheduler aufgerufen, um das Dialog-Feld von *
*                     seiner Deaktivierung in Kenntnis zu setzen.          *
* Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-   *
*                         rückgeliefert wurde.                             *
* Return-Wert       : keiner                                               *
***************************************************************************/
```

fen wurde, werden die verschiedenen Buttons automatisch von 0 an durchnumeriert. Der Return-Wert von DlgStart() spiegelt also die Nummer des ausgewählten Action-Buttons wieder.

◂◂ Listing 3: Fortsetzung

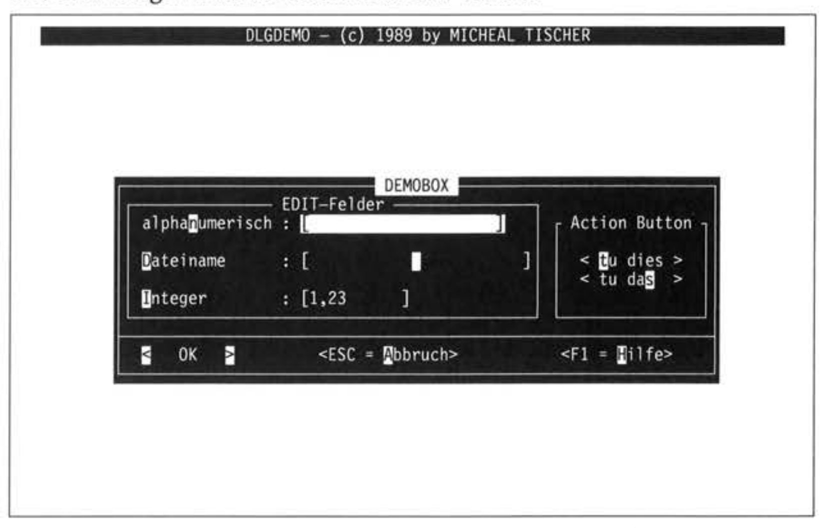


Bild 3: Das Demo-Programm DLGDEMO.C demonstriert die Erstellung einer Dialogbox aus Edit-Feldern und Action-Buttons

Null Problemo mit der Demo

Das Demo-Programm DLGDEMO.C demonstriert die Deklaration der verschiedenen Datenstrukturen, die zur Beschreibung einer Dialogbox benötigt werden und die Arbeit mit der DlgStart()-Funktion. Neben den in dieser Folge vorgestellten DLG-Modulen werden auch die Module aus den ersten beiden Folgen dieser Serie benötigt, die den Zugriff auf Maus, Tastatur und Bildschirm abwickeln.

Die verschiedenen DLG-Module sind nicht an ein bestimmtes Speichermodell gebunden und können daher an Ihre speziellen (Speicher-)Bedürfnisse angepaßt werden. Erstellen Sie das DLGDEMO-Programm unter dem Speichermodell SMALL durch folgenden Aufruf Ihres Microsoft C Compilers:

```
cl /AS dlgdmeo.c dlg.c dlgedit.c dlgab.c vio.c kbm.c
```

Das Wort zum Objekt

Abschließend noch etwas Etymologie nach dem Motto »Woher hat das Objekt seinen Namen?«.

Wer bereits mit den Ideen objektorientierter Programmierung vertraut ist, oder sich anhand des Artikels über QuickPascal in diesem *Microsoft System Journal* vertraut macht, der wird schnell feststellen, daß die Entwicklung einer Dialogverwaltung geradezu ein klassisches Anwendungsgebiet objektorientierter Programmiertechniken darstellt. Dies ist kein Zufall, denn gerade die Probleme, die es bei der Entwicklung hochinteraktiver Systeme wie SmallTalk, MS-Windows oder dem Presentation Manager zu meistern gilt, haben die Entwicklung objektorientierter Programmiersprachen forciert.

▸▸ *Listing 3: Fortsetzung*

```
void edit_deak( EDITP ep )
{
 /*— farbliche Hervorhebung des Feldes zurücknehmen ——————————————*/
 MouHideMouse();                              /* Maus-Cursor ausblenden */
 VioPrint( ep->x-1, ep->y, F(nm), FALSE, "[" );
 VioPrint( ep->x + ep->d.visi, ep->y, F(nm), FALSE, "]" );
 ep->xlinks = ep->xcur = 0;            /* Feld ab Position 0 darstellen */
 edit_print( ep, F(nm) );
 ep->maus = ep->aktiv = FALSE;                     /* nicht mehr aktiv */
 if ( ep->insert )                       /* ist der Insert-Modus aktiv? */
  edit_toggle_insert( ep );                         /* Ja, abschalten */
 edit_shrink( ep );                       /* String in alte Form bringen */
 VioHideCursor();                     /* Cursor vom Bildschirm entfernen */
 MouShowMouse();                       /* Maus-Cursor wieder einblenden */
}
/***************************************************************************
* Funktion         : e d i t _ m a u s                                    *
**_____________________________________________________________________**
* Aufgabe          : Wird vom Scheduler aufgerufen, um dem Dialog-Feld ein *
*                    Mausereignis zu übergeben                            *
* Eingabe-Parameter: EP   = der Zeiger, der beim Aufruf von edit_start zu- *
*                           rückgeliefert wurde.                          *
*                    X, Y = Position des Mauscursors realtiv zu oberen    *
*                           linken Bildschirmecke                         *
*                    EV   = Event-Maske, die das Ereignis beschreibt, um  *
*                           dessentwillen die Funktion aufgerufen wird    *
* Return-Wert      : Reaktionscode (TF_...)                               *
***************************************************************************/
BYTE edit_maus( EDITP ep, BYTE x, BYTE y, BYTE ev )
{
 if ( ep->aktiv )                          /* ist das Feld bereits aktiv? */
  {                         /* Ja, Status des linken Mausknopfs feststellen */
  if ( ev & EV_LEFT_REL )        /* wurde der linke Mausknopf losgelassen? */
   {
   ep->maus = FALSE;
   return TF_WEITER;                          /* Ja, Ereignis weiterreichen */
   }
  else                                       /* Nein, Mausposition auswerten */
   if ( ep->maus )                           /* auf Mausbewegung eingehen */
    {
    if ( x < ep->x )                              /* links vom Eingabefeld? */
     {                                                              /* Ja */
     edit_links( ep );                    /* Cursor eine Spalte nach links */
     MausPause();                       /* Programmausführung kurz anhalten */
     }
    else
     if ( x >= ep->x + ep->d.visi )               /* rechts vom Eingabefeld? */
      {                                                             /* Ja */
      edit_rechts( ep );                 /* Cursor eine Spalte nach rechts */
      MausPause();                      /* Programmausführung kurz anhalten */
      }
     else                                          /* Cursor im Eingabefeld */
      {
      ep->xcur = x - ep->x;                      /* Cursorspalte berechnen */
      VioSetCursor( ep->x + ep->xcur, ep->y );            /* Cursor setzen */
      }
    return TF_ACCEPTED;
    }
   else                                /* die Maus wird noch nicht überwacht */
    {              /* befindet sich die Maus innerhalb des Eingabefeldes? */
    if ( y==ep->y  &&  x>=ep->x  &&  x<= ep->x+ep->d.visi-1 )
     {                                                              /* Ja */
     ep->xcur = x - ep->x;               /* Cursorpos. im Eingabefeld merken */
     ep->maus = TRUE;                          /* Mausbewegung überwachen */
     return TF_ACCEPTED;
     }
    else                                  /* Maus außerhalb des Eingabefeldes */
     return TF_WEITER;
    }
  }
 else                              /* Nein, testen ob es jetzt aktiviert wird */
  {
  if ( ( ev & EV_LEFT_PRESS )    &&            /* linker Maus-Button muß    */
       y == ep->y                &&            /* niedergedrückt sein und   */
       x >= ep->x                &&            /* der Maus-Cursor muß sich  */
       x <= ep->x + ep->d.visi -1 &&           /* im Eingabefeld befinden   */
       ep->d.enabled              )            /* und Feld muß enabled sein */
   {
   ep->xcur = x - ep->x;                 /* Cursorpos. im Eingabefeld merken */
   ep->maus = TRUE;                            /* Mausbewegung überwachen */
   return TF_MAUS;
   }
  else           /* obige Bedingung wird nicht erfüllt, Event weiterreichen */
   return TF_WEITER;
  }
}

/***************************************************************************
* Funktion         : e d i t _ c a n                                      *
**_____________________________________________________________________**
* Aufgabe          : Wird vom Scheduler aufgerufen, um festzustellen, ob   *
*                    das Dialogfeld bereit ist, aktiviert zu werden.       *
* Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-   *
*                         rückgeliefert wurde.                            *
* Return-Wert      : TRUE, wenn das Dialog-Feld aktiviert werden kann,    *
*                    sonst FALSE                                          *
***************************************************************************/
BOOL edit_can( EDITP ep )
{
 return ep->d.enabled;        /* ENABLE-Flag aus EDIT-Struktur zurückliefern */
}
/***************************************************************************
* Funktion         : e d i t _ a k t i v                                  *
**_____________________________________________________________________**
* Aufgabe          : Wird vom Scheduler aufgerufen, um das Dialog-Feld von *
*                    seiner Aktivierung in Kenntnis zu setzen.            *
* Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von edit_start zu-   *
*                         rückgeliefert wurde.                            *
*                    WHY= warum wurde das Feld aktiviert.                  *
* Return-Wert      : keiner                                               *
* Info             : Der WHY-Parameter enthält eine der TF_...-Konstanten  *
*                    wie TF_FELD_VOR, TF_MAUS etc.                        *
***************************************************************************/
void edit_aktiv( EDITP ep, BYTE why )
{
 /*— Dialog-Feld farblich hervorheben, Cursor anzeigen ————————————*/
 if ( !ep->aktiv )                         /* ist das Feld bereits aktiv? */
  {                                                               /* Nein */
  MouHideMouse();                              /* Maus-Cursor ausblenden */
  if ( why != TF_MAUS )                      /* Aktivierung durch Maus? */
   ep->xcur = 0;                        /* Nein, Cursor an linken Rand */
  VioPrint( ep->x-1, ep->y, F(hi), FALSE, "[" );
  VioPrint( ep->x + ep->d.visi, ep->y, F(hi), FALSE, "]" );
  ep->xlinks = 0;               /* Cursor steht am linken Rand des E-Feldes */
  edit_fill( ep );                            /* Eingabesgtring auffüllen */
  edit_print( ep, F(hi) );                      /* Eingabestring ausgeben */
  VioSetCursor( ep->x + ep->xcur, ep->y );
  ep->aktiv = TRUE;                                /* Feld ist jetzt aktiv */
  MouShowMouse();                      /* Maus-Cursor wieder einblenden */
  }
}
```

Objektorientierte Programmiertechniken, etwa die Zusammenbindung von Daten und Programmcode (Data Encapsulation) oder die Vererbung von »Attributen« (Inheritance), ließen sich wunderbar bei der Realisation einer Dialog-Verwaltung einsetzen. Und tatsächlich gleicht die Realisation der hier vorgestellten Dialogverwaltung sehr stark dem Programmcode, den beispielsweise der C++-Precompiler von Glockenspiel bei der Umsetzung von C++-Programmen in Standard-C-Code erzeugt.

Wie aber würde der hier vorgestellte Dialog-Manager in einem objektorientierten C-Dialog, beispielsweise in C++, aussehen?

Am Dialog-Manager selbst würde sich wohl wenig ändern, mehr dafür aber an der Repräsentation der einzelnen Objekte, die nun tatsächlich als Objekte im Sinne der Zusammenbindung von Programmcode und Daten formuliert werden würden. Welche Daten ein solches Objekt beinhaltet, wäre dabei für den Dialog-Manager nicht von Bedeutung, denn er kommt mit den Daten (auch in der nicht objektorientierten Version) gar nicht in Berührung. Wichtig für ihn ist lediglich, daß jedes Objekt bestimmte Methods (die bisherigen Dialogfunktion) bereitstellt, über die er mit dem Objekt kommunizieren kann. Da jeder dieser Methods automatisch als implizites Argument ein Zeiger auf die adressierte Instanz des Objekt-Typs übergeben wird, würde die Übergabe eines Zeigers, wie wir sie in den jetzigen Dialogfunktionen benötigen, entfallen.

Hier würde der Compiler dem Programmierer also eine Menge Verwaltungsarbeit abnehmen und sich der Dialog-Manager und die einzelnen Objekte dadurch nicht nur übersichtlicher gestalten lassen, sondern auch die Fehleranfälligkeit reduzieren.

Aber auch in anderer Hinsicht können die Ideen des objektorientierten Programmierens bei der Entwicklung einer Dialog-Verwaltung helfen. Dann nämlich, wenn es gilt, aus den einfachen Grundobjekten komplexere Objekte aufzubauen. Leicht ließe sich aus einem Edit-Feld z.B. ein numerisches Eingabefeld gestalten, indem das Num-Objekt mittels Vererbung aus dem Edit-Objekt hervorgeht, es dabei aber ein wenig modifiziert wird. Zunächst müßten die Variablen innerhalb des Objekts um eine Variable zur Aufnahme eines numerischen Wertes ergänzt werden. Gleichzeitig werden Start- und Ende-Method so erweitert, daß die Zahl beim Aufruf der Start-Method zunächst nach ASCII umgewandelt und das Ergebnis dieser Umwandlung dann in den Eingabe-String eingetragen wird. Dessen Inhalt wird beim Schließen der Dialogbox, und dem damit verbundenen Aufruf der Ende-Method, später wieder in eine Zahl umgerechnet und in die Zahl-Variable eingetragen.

Wie die vorliegende Dialog-Verwaltung zeigt, können diese Aufgaben auch mit tradierten Programmiertechniken bewältigt werden, doch will

dies nichts bedeuten, denn letztendlich kann man bei entsprechender Geduld auch die komplexesten mathematischen Berechnungen mit einem Abakus durchführen. Doch wer käme heute noch auf diese Idee?

Michael Tischer

Listing 4: DLGAB.C

```
/***************************************************************************/
/*                              D L G A B . C                              */
/*-------------------------------------------------------------------------*/
/*    Aufgabe        : Enthält die Funktionen zur Verwaltung von Action-   */
/*                     Buttons, die innerhalb von Dialog-Boxen zum Einsatz */
/*                     kommen.                                             */
/*                     Dieses Modul muß in Verbindumg mit dem DLG-Modul zum */
/*                     Einsatz kommen.                                     */
/*-------------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                     */
/*    entwickelt am  : 13.07.1989                                          */
/*    letztes Update : 17.07.1989                                          */
/*-------------------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] DLGAB.C /C                         */
/*                     dann mit dem DLG-Modul und anderen Moduln verbinden */
/***************************************************************************/

/*-- Include-Dateien einbinden -------------------------------------------*/

#include <malloc.h>
#include <string.h>
#include "dlg.h"

/*-- interne Datenstruktur des Dialogtyps AB (Action Button )-------------*/

typedef struct    /* die Daten werden für jeden Action-Button festgehalten -*/
 {
  TASTE hotkey;          /* Hotkey des Action Buttons (NOKEY = kein Hotkey) */
  BYTE  x1, x2,                              /* Start- und Endspalte des ABs */
        y,                                                        /* Zeile */
        xcur;                              /* Spalte für blinkenden Cursor */
 } ABINTERN;

typedef ABINTERN *ABIP;                 /* Zeiger auf eine interne AP-Struktur */

/*-- Prototypen für Funktionen, die in diesem Modul deklariert und in der --*/
/*-- globalen Variablen std_ab_fkt zusammengefaßt werden                 --*/

void * ab_start ( ABGROUP * dptr );
void   ab_newval( ABIP abip, void * dptr );
void   ab_ende  ( ABIP abip );
BYTE   ab_taste ( ABIP abip, TASTE key );
void   ab_deak  ( ABIP abip );
void   ab_aktiv ( ABIP abip, BYTE why );
BOOL   ab_maus  ( ABIP abip, BYTE x, BYTE y, BYTE ev );
BOOL   ab_can   ( ABIP abip );

/*-- globale Variablen, öffentlich ---------------------------------------*/

DLGFKT std_ab_fkt =                       /* Standard Dialog-Funktionen für */
 {                                         /* eine Gruppe von Action-Buttons */
   ab_start, ab_newval, ab_ende , ab_taste,
   ab_deak,  ab_maus,   ab_aktiv, ab_can
 };

/*-- globale Variablen, modulintern --------------------------------------*/

static ABGROUP   abg;        /* dient den Funktionen für die Action-Buttons */
static BYTE      aktab;                                    /* aktueller AB */
static BOOL      abaktiv;                 /* ist TRUE, wenn innerhalb ABs */
static BOOL      stdinvert;      /* ist TRUE, wenn Standard-AB invertiert */

/***************************************************************************
* A B            Es folgen die verschiedenen Funktionen des Dialogtyps AB  *
*                ( eine Gruppe von Action Buttons )                        *
***************************************************************************/

/***************************************************************************
*  Funktion        : a b _ t e s t k e y                                   *
**-----------------------------------------------------------------------**
*  Aufgabe         : Stellt fest, ob die übergebene Taste einem der Action- *
*                    Buttons zugeordnet wurde, oder dessen Hotkey ent-     *
*                    spricht.                                              *
*  Eingabe-Parameter: ABIP = Zeiger auf internen AB-Vektor                 *
*                    KEY  = Taste                                          *
*  Return-Wert     : -1, wenn kein zugehöriger Action Button gefunden wurde,*
*                    sonst die Nummer des Action-Buttons                   *
*  Info            : Diese Funktion findet nur intern Verwendung, sie wird *
*                    nicht vom Scheduler aufgerufen.                       *
***************************************************************************/

int ab_testkey( ABIP abip, TASTE key )
{
 BYTE  i;                                              /* Schleifenzähler */
 ABPTR labptr;                       /* Laufzeiger in Vektor mit AB-Daten */

 /*-- den Vektor mit den AB-Beschreibern durchlaufen ---------------------*/
 for ( i = 0, labptr = abg.abp;
       i < abg.anz  &&  !(labptr->key == key  &&  labptr->enabled);
       ++i, ++labptr )
  ;
 if ( i == abg.anz )                          /* Taste bereits entdeckt? */
  {                                  /* Nein, jetzt die Hotkeys untersuchen */
   for ( i = 0; i < abg.anz  &&  abip->hotkey != key; ++i, ++abip )
    ;
  }
 return ( i < abg.anz ) ? i : -1;               /* i < abg.anz : gefunden! */
}

/***************************************************************************
*  Funktion        : a b _ m a r k                                         *
**-----------------------------------------------------------------------**
*  Aufgabe         : Markiert einen Action-Button als gewählt oder nicht   *
*                    gewählt.                                              *
*  Eingabe-Parameter: ABIP = Zeiger auf internen AB-Vektor                 *
*                    NR   = Nummer des angesprochenen Action-Buttons       *
*                    MARK = TRUE, wenn es sich um den aktuellen Action-    *
*                           Button handelt                                 *
*  Return-Wert     : keiner                                                *
*  Info            : - Mit dieser Funktion können nur Action-Button bear-  *
*                      beitet werden, die enabled sind.                    *
*                    - Diese Funktion findet nur intern Verwendung, sie    *
*                      wird nicht vom Scheduler aufgerufen.                *
***************************************************************************/

void ab_mark( ABIP abip, BYTE nr, BOOL mark )
{
 register BYTE f,                                          /* Ausgabefarbe */
               s,                                         /* Ausgabespalte */
               z;                                          /* Ausgabezeile */

 MouHideMouse();                                  /* Maus-Cursor ausblenden */
 abip += nr;                  /* den Beschreiber des angespr. ABs adressieren */
 VioPrint( s = abip->x1, z = abip->y, f = mark ? F(hi) : F(nm), FALSE, "<" );
 VioPrint( abip->x2, z, f, FALSE, ">" );
 DlgPrint( s+1, z, (abg.abp+nr)->name, F(nm), F(hk) );
 if ( mark )                         /* aktuellen Action Button markiert? */
  VioSetCursor( abip->xcur, z );                /* Ja, Cursor darauf setzen */
 if ( nr == abg.standard )           /* wurde der Standard-AB gezeichnet? */
  stdinvert = FALSE;                 /* Ja, Standard-AB nicht mehr invers */
 MouShowMouse();                           /* Maus-Cursor wieder einblenden */
}
```

```
/***************************************************************************
*  Funktion        : a b _ i n v e r t                                     *
**-----------------------------------------------------------------------**
*  Aufgabe         : Invertiert einen Action-Button                        *
*  Eingabe-Parameter: ABIP = Zeiger auf internen AB-Vektor                 *
*                    NR   = Nummer des angesprochenen Action-Buttons       *
*  Return-Wert     : keiner                                                *
*  Info            : - Mit dieser Funktion können nur Action-Button bear-  *
*                      beitet werden, die enabled sind.                    *
*                    - Diese Funktion findet nur intern Verwendung, sie    *
*                      wird nicht vom Scheduler aufgerufen.                *
***************************************************************************/
void ab_invert( ABIP abip, BYTE nr )
{
 abip += nr;                 /* den Beschreiber des angespr. ABs adressieren */

 MouHideMouse();                                  /* Maus-Cursor ausblenden */
 VioColor( abip->x1, abip->y, abip->x2, abip->y, F(hi) );
 VioSetCursor( abip->xcur, abip->y );                       /* Cursor setzen */
 MouShowMouse();                           /* Maus-Cursor wieder einblenden */
 if ( nr == abg.standard )            /* wurde der Standard-AB gezeichnet? */
  stdinvert = TRUE;                      /* Ja, Standard-AB ist jetzt invers */
}
/***************************************************************************
*  Funktion        : a b _ s t a r t                                       *
**-----------------------------------------------------------------------**
*  Aufgabe         : Wird vom Scheduler während der Initialisierung einer  *
*                    Dialogbox für die Gruppe der Action Buttons aufgerufen.*
*  Eingabe-Parameter: DPTR = Zeiger auf den Datenblock des Actions Buttons *
*  Return-Wert     : Ein Zeiger, den der Scheduler bei allen folgenden Auf- *
*                    rufen den Funktionen abaktiv, abtast etc. übergibt.   *
***************************************************************************/
void * ab_start ( ABGROUP * dptr )
{
 BYTE   i,                                             /* Schleifenzähler */
        f;                                                /* Ausgabefarbe */
 char * sptr;                   /* Zeiger auf den Namen des Action-Buttons */
 ABIP   abip,          /* Zeiger auf allokierten Vektor mit internen Infos */
        labip;                               /* Laufzeiger in obigen Vektor */
 ABPTR  labptr;                        /* Laufzeiger auf einen AB-Beschreiber */

 abg = *dptr;        /* übergebene Struktur in globale Variable ABG kopieren */

 /*-- die einzelnen Action-Buttons aufbauen, Hotkeys und Pos. merken ------*/

 labip = abip = (ABIP) malloc( sizeof(ABINTERN) * abg.anz );
 for ( labptr = abg.abp, i = abg.anz; i ; --i, ++labip, ++labptr )
  {                              /* den AB, auf den LABPTR zeigt, bearbeiten */
   labip->x2 = VL( labptr->x + strlen( labptr->name ) + 1 );
   labip->hotkey = DlgPrint( (labip->x1 = VL(labptr->x) ) + 1,
                             labip->y = VO(labptr->y),
                             sptr = labptr->name,
                             f = labptr->enabled? F(nm) : F(da),
                             labptr->enabled? F(hk) : F(dk)        );

   if ( !labptr->enabled )                          /* ist der AB disabled? */
    labip->hotkey = NOKEY;                      /* Ja, es gibt keinen Hotkey */

   if ( labip->hotkey != NOKEY)                     /* gibt es einen Hotkey? */
    {                                                               /* Ja */
     --labip->x2;                  /* Hotkey-Zeichen nicht in Länge mitzählen */
     labip->xcur = labip->x1 + 1 + ( strchr( sptr, HOTKEY ) - sptr );
    }
   else                                             /* es gibt keinen Hotkey */
    {                            /* erstes nicht-Space im Namen des ABs suchen */
     for ( sptr = labptr->name; *sptr == ' '; ++sptr )
      ;
     labip->xcur = labip->x1 + 1 + ( sptr - labptr->name );
    }

   /*-- Begrenzer ausgeben -----------------------------------------------*/
   VioPrint( labip->x1, labip->y, f, FALSE, "<" );
   VioPrint( labip->x2, labip->y, f, FALSE, ">" );
  }

 ab_mark( abip, aktab = abg.standard, TRUE );      /* akt. Button markieren */

 abaktiv = FALSE;                             /* ABs sind noch nicht aktiv */
 return abip;                          /* Zeiger an Scheduler zurückliefern */
}
/***************************************************************************
*  Funktion        : a b _ n e w v a l                                     *
**-----------------------------------------------------------------------**
*  Aufgabe         : Wird nicht direkt vom Scheduler, sondern von einer mit *
*                    ihm zusammenarbeitenden Interaktions-Fukntion aufge-  *
*                    rufen, wenn sich der Inhalt eines EDIT-Feldes durch   *
*                    äußeres Ereignis verändert hat.                       *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von ab_start zurück- *
*                          geliefert wurde.                                *
*  Return-Wert     : keiner                                                *
***************************************************************************/
void ab_newval( ABIP abip, void * dptr )
{
}
/***************************************************************************
*  Funktion        : a b _ e n d e                                         *
**-----------------------------------------------------------------------**
*  Aufgabe         : Wird vom Scheduler während des Schließens des Dialog- *
*                    box aufgerufen, um EDIT Gelgegenheit zu geben, Clean- *
*                    Up-Arbeiten durchzuführen.                            *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von ab_start zurück- *
*                          geliefert wurde.                                *
*  Return-Wert     : keiner                                                *
***************************************************************************/

void ab_ende( ABIP abip )
{
 free( abip );                      /* den allokierten Vektor wieder freigeben */
}
/***************************************************************************
*  Funktion        : a b _ t a s t e                                       *
**-----------------------------------------------------------------------**
*  Aufgabe         : Wird vom Scheduler aufgerufen, um dem Dialog-Feld eine *
*                    Taste zu übergeben.                                   *
*  Eingabe-Parameter: EP    = der Zeiger, der beim Aufruf von ab_start zu- *
*                             rückgeliefert wurde.                         *
*                    TASTE = Code der zu bearbeitenden Taste               *
*  Return-Wert     : Reaktionscode (TF_...)                                *
***************************************************************************/
```

Listing 4: Fortsetzung und Ende

```c
BYTE ab_taste( ABIP abip, TASTE key )
{
 ABIP labip;                       /* Laufzeiger in den internen AB-Vektor */
 int  retcode,                                  /* nimmt Return-Code auf */
      neuakt;                                     /* neuer aktueller AB */

 if ( abaktiv )                                  /* sind die ABs aktiv? */
  {                                                             /* Ja */
   switch ( key )                                   /* Taste auswerten */
    {
     case ' '   :   /*——————— SPACE wählt aktuellen Action Button aus -*/
       labip = abip + aktab;    /* Zeiger auf interne Infos zu aktuellen AB */
       DlgPrint(labip->x1+1, labip->y, (abg.abp+aktab)->name, F(hi), F(hk));
       return aktab;      /* Terminationscode ist Nummer des Action Buttons */

     case TAB   :   /*———————— TAB führt zum nächsten Action-Button -*/
       for ( neuakt = aktab ;
             ++neuakt != abg.anz  &&  !(abg.abp+neuakt)->enabled; )
        ;
       if ( neuakt == abg.anz )              /* auf letztem Action-Button? */
        return TF_FELD_VOR;          /* Ja, ins nächste Dialog-Feld springen */
       else                  /* Nein, auf nächsten Action-Button umschalten */
        {
         ab_mark( abip, aktab, FALSE );          /* aktuellen AB ausblenden */
         ab_mark( abip, aktab = neuakt, TRUE );      /* neuen AB einblenden */
         return TF_ACCEPTED;                     /* Taste wurde verarbeitet */
        }

     case BACKTAB   :   /*- BACKTAB führt zum vorhergehenden Action-Button */
       for ( neuakt = aktab;
             --neuakt != -1  &&  !(abg.abp+neuakt)->enabled ; )
        ;
       if ( neuakt != -1 )                    /* auf erstem Action-Button? */
        {             /* Nein, auf vorhergehenden Action-Button umschalten */
         ab_mark( abip, aktab, FALSE );          /* aktuellen AB ausblenden */
         ab_mark( abip, aktab = neuakt, TRUE );      /* neuen AB einblenden */
         return TF_ACCEPTED;                     /* Taste wurde verarbeitet */
        }
       else                /* Ja, zum vorhergehenden Dialog-Feld springen */
        return TF_FELD_RUECK;

     case CR        : /*——————————————————————————————————— RETURN */
       return (( retcode = ab_testkey( abip, key ) ) != -1 ) ? retcode
                                                                : aktab;

     default :      /*—————————————————————————— jede andere Taste -*/
       return (( retcode = ab_testkey( abip, key ) ) != -1 ) ? retcode
                                                                :  TF_WEITER;
    }
  }
 else                                 /* Action Button sind noch nicht aktiv */
  if ( key == CR )                              /* wurde RETURN betätigt? */
   return abg.standard;             /* Ja, RETURN wählt den Standard-AB aus */
  else                       /* Nein, ist ein AB mit dieser Taste verbunden? */
  return (( retcode = ab_testkey( abip, key ) ) != -1 ) ? retcode
                                                        :  TF_WEITER;
}
/***************************************************************************
*  Funktion          : a b _ d e a k                                       *
**_____________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler aufgerufen, um das Dialog-Feld von *
*                      seiner Deaktivierung in Kenntnis zu setzen.         *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von ab_start zurück- *
*                            geliefert wurde.                              *
*  Return-Wert       : keiner                                              *
***************************************************************************/
void ab_deak( ABIP abip )
{
 /*— den Standard-Button aktivieren ——————————————————————————*/

 if ( aktab != abg.standard )                /* ist der bereits ausgewählt? */
  {                          /* Nein, alten ausblenden, neuen einblenden */
   ab_mark( abip, aktab, FALSE );
   ab_mark( abip, aktab = abg.standard , TRUE );
  }
 abaktiv = FALSE;                                   /* ABs nicht mehr aktiv */
 VioHideCursor();                           /* Cursor vom Bildschirm entfernen */
}
/***************************************************************************
*  Funktion          : a b _ m a u s                                       *
**_____________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler aufgerufen, um dem Dialog-Feld ein *
*                      Mausereignis zu übergeben                           *
*  Eingabe-Parameter: EP    = der Zeiger, der beim Aufruf von edit_start zu- *
*                               rückgeliefert wurde.                       *
*                      X, Y  = Position des Mauscursors realtiv zu oberen  *
*                               linken Bildschirmecke                      *
*                      EV    = Event-Maske, die das Ereignis beschreibt, um *
*                               dessentwillen die Funktion aufgerufen wird *
*  Return-Wert       : Reaktionscode (TF_...)                              *
***************************************************************************/
BYTE ab_maus  ( ABIP abip, BYTE x, BYTE y, BYTE ev )
{
 BYTE  i;                                              /* Schleifenzähler */
 ABPTR labptr;                 /* Laufzeiger auf die einzelnen AB-Beschreiber */
 ABIP  labip;             /* Laufzeiger in Vektor mit internen Informationen */

 if ( abaktiv )                           /* sind die Action-Buttons aktiv? */
  {                                                              /* Ja */
   if ( ev & EV_LEFT_REL )      /* wurde der linke Mausbutton losgelassen? */
    {            /* Ja, feststellen, ob sich die Maus über einem AB befand */
     for ( i = 0, labptr = abg.abp; i < abg.anz ; ++i, ++abip, ++labptr )
      if ( labptr->enabled  &&  y==abip->y  &&  x>=abip->x1  &&  x<=abip->x2 )
       return i;                                  /* Ja, Dialog-Box beenden */
     return TF_WEITER;                                /* Maus nicht über AB */
    }
   else                        /* der linke Mausknopf ist noch niedergedrückt */
    {    /* feststellen, ob sich die Maus noch über dem aktiven AB befindet */
     labip = abip + aktab;   /* Zeiger auf internen Beschreiber des akt. AB */
     if ( y!=labip->y  || x<labip->x1  ||  x>labip->x2 )
      {              /* Maus ist jetzt nicht mehr auf dem aktuellen Button */
       /*— testen, ob sich die Maus über einem anderen AB befindet———— */

       for ( i = 0, labptr = abg.abp, labip = abip;
             i < abg.anz ;
             ++i, ++labip, ++labptr )
        if ( labptr->enabled  &&  y==labip->y  &&
             x>=labip->x1  &&  x<=labip->x2 )
         break;                                     /* Maus über anderem AB */

       if ( i != abg.anz )                    /* Maus über einem anderen AB? */
        {                                                            /* Ja */
         ab_mark( abip, aktab, FALSE );           /* aktuellen AB ausblenden */
         ab_invert( abip, aktab = i );              /* Ja, neuen invertieren */
        }
       else                       /* Nein, Standard-AB auswählen und markieren */
        {
         if ( aktab != abg.standard )               /* bereits auf Standard-AB */
          {                                                         /* Nein */
           ab_mark( abip, aktab, FALSE );        /* aktuellen AB ausblenden */
           ab_mark( abip, aktab = abg.standard, TRUE );
          }
         else
          if ( stdinvert )
           ab_mark( abip, aktab, TRUE );
        }
      }
     else                                        /* noch auf dem gleichen Button */
      if ( aktab == abg.standard  &&  !stdinvert )
       ab_invert( abip, aktab );            /* Std-AB war noch nicht invertiert */

     return TF_ACCEPTED;                             /* Event wurde verarbeitet */
    }
  }
 else                                          /* Nein, die ABs sind nicht aktiv */
  if  ( ev & EV_LEFT_PRESS )  /* wurde der linke Mausbutton niedergedrückt? */
   {                   /* ja, testen, ob sich die Maus über einem AB befindet */
    for ( i = 0, labptr = abg.abp; i < abg.anz ; ++i, ++abip, ++labptr )
     if ( labptr->enabled  &&  y==abip->y  &&  x>=abip->x1  &&  x<=abip->x2 )
      {                            /* Maus befindet sich über einem enableden AB! */
       aktab = i;                                       /* Nummer des ABs merken */
       return TF_MAUS;                                          /* AB aktivieren */
      }
    return TF_WEITER;                           /* Maus nicht über einem der ABs */
   }
  else                                                                  /* Nein */
   return TF_WEITER;
}
/***************************************************************************
*  Funktion          : a b _ c a n                                         *
**_____________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler aufgerufen, um festzustellen, ob  *
*                      das Dialogfeld bereit ist, aktiviert zu werden.     *
*  Eingabe-Parameter: EP = der Zeiger, der beim Aufruf von ab_start zurück- *
*                            geliefert wurde.                              *
*  Return-Wert       : TRUE, wenn das Dialog-Feld aktiviert werden kann,   *
*                      sonst FALSE                                         *
***************************************************************************/
BOOL ab_can( ABIP abip )
{
 return TRUE;                                   /* ABs können aktiviert werden */
}
/***************************************************************************
*  Funktion          : a b _ a k t i v                                     *
**_____________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler aufgerufen, um das Dialog-Feld von *
*                      seiner Aktivierung in Kenntnis zu setzen.           *
*  Eingabe-Parameter: EP  = der Zeiger, der beim Aufruf von ab_start zurück- *
*                             geliefert wurde.                             *
*                      WHY = warum wird das Feld aktiviert                 *
*  Return-Wert       : keiner                                              *
***************************************************************************/
void ab_aktiv( ABIP abip, BYTE why )
{
 BYTE neuab;                    /* Nummer des neuen aktuellen Action Buttons */

 abaktiv = TRUE;                                        /* ABs jetzt aktiv */

 /*— Wurde der Aufruf durch eine TF_FELD_VOR- oder TF_FELD_RUECK-Message —*/
 /*— herbeigeführt, muß ein aktiver Action-Button ausgewählt werden. Bei —*/
 /*— allen anderen Messages (TF_AKTIV etc.) wird davon ausgegangen, daß  —*/
 /*— bereits ein aktiver Action Button gewählt und seine Nummer in der   —*/
 /*— Variablen aktab gespeichert wurde                                   —*/

 neuab = aktab;       /* davon ausgehen, daß neuer AB bereits gewählt wurde */
 switch ( why )                              /* Grund des Aufrufs untersuchen */
  {
   case TF_FELD_VOR   :                                       /* ein Feld vor */
     for ( neuab = 0; !(abg.abp+neuab)->enabled; ++neuab )
       ;
     break;

   case TF_FELD_RUECK :                                    /* ein Feld zurück */
     for ( neuab = abg.anz-1; !(abg.abp+neuab)->enabled; --neuab )
       ;
     break;

   case TF_MAUS       :                            /* aktivierung durch Maus */
     if ( aktab != abg.standard )        /* wird der Standard-AB aktiviert? */
      ab_mark( abip, abg.standard, FALSE );          /* Nein, alten AB ausbl. */
     ab_invert( abip, aktab );                         /* neuen AB invertieren */
     break;
  }
 if ( neuab != aktab )                            /* wurde ein neuer AB gewählt? */
  {                              /* Ja, alten ausblenden, neuen einblenden */
   ab_mark( abip, aktab, FALSE );
   ab_mark( abip, aktab = neuab, TRUE );
  }
 else                                         /* Nein, nur Cursor in AB setzen */
   VioSetCursor( (abip+aktab)->xcur, (abip+aktab)->y );
}
```

►► Listing 5: DLGDEMO.C

```c
/****************************************************************************/
/*                            D L G D E M O. C                              */
/*__________________________________________________________________________*/
/*    Aufgabe        : Demonstriert die Erstellung und Verwaltung von Dia-  */
/*                      logboxen durch die Module DLG, DLGEDIT und DLGAB.   */
/*__________________________________________________________________________*/
/*    Autor          : MICHAEL TISCHER                                      */
/*    entwickelt am  : 13.07.1989                                           */
/*    letztes Update : 17.07.1989                                           */
/*__________________________________________________________________________*/
/*    Erstellung     : CL /A[S|M|C|L|H] DLGDEMO.C DLG.C DLGAB.C DLGEDIT.C   */
/*                                       VIO.C KBM.C                        */
/****************************************************************************/
/*— Include-Dateien einbinden ————————————————————————————————————*/

#include "dlg.h"

/*— Forward-Funktionsdeklarationen ———————————————————————————————*/
void paint( void );

DLGCOL demofarbe =                  /* Farben in der Dialogox für Color-Modus */
 {
  COL( WEISS, CYAN ),                             /* Rahmenfarbe              */
  COL( SCHWARZ, CYAN ),                           /* normale Zeichen          */
  COL( GELB, SCHWARZ ),                           /* hervorgehobene Zeichen   */
  COL( ROT, CYAN ),                               /* Hotkeys                  */
  COL( HGRAU, CYAN ),                             /* inaktive Felder          */
  COL( HGRAU, CYAN )                              /* inaktive Felder, hotkey  */
 };

/*— Daten für Eingabefeld #1 —————————————————————————————————————*/
BYTE flbuf[21] = "";                                      /* Eingabepuffer */
EMENGE flmenge;                                           /* Eingabemenge */

EDITFELD tb_dlgfl =
 {
  TRUE, 3, 2, 20, 20, "alpha#numerisch : ", flbuf, flmenge
 };
```

```
/*— Daten für Eingabefeld #2 ———————————————————————————————*/

BYTE f2buf[81] = "";                                /* Eingabepuffer */
EMENGE f2menge;                                     /* Eingabemenge */

EDITFELD tb_dlgf2 =
 {
  TRUE, 3, 4, 80, 23, "#Dateiname      : ", f2buf, f2menge
 };

/*— Daten für Eingabefeld #3 ———————————————————————————————*/

BYTE f3buf[11] = "1,23";                            /* Eingabepuffer */
EMENGE f3menge;                                     /* Eingabemenge */

EDITFELD tb_dlgf3 =
 {
   TRUE, 3, 6, 10, 10, "#Integer        : ", f3buf, f3menge
 };


/*— Daten für die Action-Buttons——————————————————————————— */

ONEAB tb_absdef[] =
 {
  { TRUE,  3, 9, "   OK   ",    NOKEY },
  { TRUE, 22, 9, "ESC = #Abbruch", ESC },
  { TRUE, 48, 9, "F1 = #Hilfe",    F1 },
  { TRUE, 50, 4, " #tu dies ",    NOKEY },
  { TRUE, 50, 5, " tu da#s  ",    NOKEY }
 };

ABGROUP tb_abs = { 5, 0, tb_absdef };

/*— Beschreibung der einzelnen Dialogfelder ————————————————*/
DLGDATEN tb_dfvek[] =
 {
  EDIT(tb_dlgf1),                  /* die Reihenfolge bestimmt auch die  */
  EDIT(tb_dlgf2),                  /* Reihenfolge, mit der die einzelnen */
  EDIT(tb_dlgf3),                  /* Felder mit TAB und mit SHIFT TAB   */
  ACTBUT(tb_abs)                   /* angesprungen werden können         */
 };

/*— Beschreibung der Dialogbox —————————————————————————————*/
DIGDES testbox = { 8, 8, 65, 11, 4, paint, tb_dfvek };

/*************************************************************************
*  Funktion          : p a i n t                                         *
**_____________________________________________________________________**
*  Aufgabe           : Wird von der Dialog-Prozedur DlgStart während des Auf- *
*                      baus der Demo-Dialogbox aufgerufen.               *
*  Eingabe-Parameter: keine                                              *
*  Return-Wert       : keiner                                            *
*************************************************************************/
void paint( void )
{
 VioPrint( VL(28), VO(0), F(hi), FALSE, " DEMOBOX " );    /* Titel ausgeben */
 VioPrintf( VL(0), VU(-2), F(dlgbox), FALSE, "├%s┤",
            VioStrep( '─', VR(-1) - VL(1) + 1 ) );
 DlgBox( 1, 1, 45, 7, EINRA, " EDIT-Felder ", FALSE );
 DlgBox( 47, 2, 17, 6, EINRA, " Action Button ", FALSE );
}

/*======================================================================*/
/*== Hauptprogramm =====================================================*/
/*======================================================================*/
```

```
void main( void )
{
 BYTE    auswahl;           /* nimmt Nr. des ausgewählten Action-Buttons auf */
 BEREICH bereich;             /* Koordinaten des OK-Buttons im Hilfe-Fenster */
 BOOL    ende;

 VioInit();                        /* die SAA-Modul VIO und KBM initialisieren */
 KbmInit();

 SetMengeAN( f1menge );              /* die drei Eingabemengen initialisieren  */
 SetMengeDatei( f2menge );
 SetMengeInt( f3menge );

 if ( VioIsColor() )                    /* ist ein Color-Adapter angeschlossen? */
  dlgcol = demofarbe;      /* Ja, Farbeinstellungen für Dialogbox vornehmen */
 VioClearScreen( VioIsColor() ? COL( WEISS, BLAU ) : NORMAL );
 VioClear( 0, 0, anzcol-1, 0, VioIsColor() ? COL( WEISS, ROT ) : INVERS );
 VioPrint( 22, 0, VioIsColor() ? COL( WEISS, ROT ) : INVERS , FALSE,
          "DLGDEMO - (c) 1989 by MICHEAL TISCHER");
 MouShowMouse();                                    /* Maus-Cursor anzeigen */
 do                                                      /* Eingabeschleife */
  {
   auswahl = DlgStart( &testbox ); /* Dlgbox aufb., Eingaben entgegennehmen */
   if ( auswahl == 2 )                               /* Hilfetaste betätigt? */
    {                                                                /* Ja */
     /*— Hilfe-Fenster öffnen und aufbauen ————————————————————*/

     MouHideMouse();                               /* Maus-Cursor ausblenden */
     VioWinOpen( 40, 16, 73, 24 );
     VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, F(dlgbox) );
     VioClear( VL(1), VO(1), VR(-1), VU(-1), F(nm) );
     VioPrint( VL(13), VO(0), F(hi), FALSE, " Hilfe! " );
     VioPrint( VL(7), VO(2), F(nm), FALSE, "Bitte \x11─┘ betätigen" );
     VioFrame( VL(21), VO(5), VR(-3), VU(-1), EINRA, F(dlgbox) );
     VioPrint( VL(25), VO(6), F(nm), FALSE, "OK" );
     MouPushPara();                                /* Mausparameter sichern */
     bereich.x1 = VL(21);                   /* Position des OK-Felds setzen */
     bereich.y1 = VO( 5);
     bereich.x2 = VR(-3);
     bereich.y2 = VU(-1);
     bereich.ptr_mask = MouPtrMask( PTRDIFCHAR( 251 ), PTRDIFCOLB( 0x70 ) );
     MouSetBereich( 1, &bereich );                /* das OK-Feld definieren */
     ende = FALSE;
     MouShowMouse();
     do
      {                                                 /* auf Event warten */
       if ( KbmEventWait( EV_LEFT_PRESS | EV_KEY_AVAIL ) & EV_KEY_AVAIL )
        ende = ( KbdGetKey() == CR );            /* Taste. Ist es Return? */
       else                               /* linker Mausknopf niedergedrückt */
        if ( MouGetBereich() == 0 )                          /* im OK-Feld? */
         {                         /* Ja, auf Loslassen des Buttons warten */
          KbmEventWait( EV_LEFT_REL );
          ende = TRUE;
         }
      }
     while ( !ende );
     MouPopPara();                         /* alte Mausparameter restaurieren */
     MouHideMouse();                               /* Maus-Cursor ausblenden */
     VioWinClose( TRUE );
     MouShowMouse();
    }
  }
 while ( auswahl!=0  &&  auswahl!=1 );
 MouHideMouse();                            /* Maus-Cursor wieder ausblenden */
 VioSetCursor( 0, 0 );             /* Cursor in obere linke Bildschirmecke */
}
```

Listing 5: Ende

Strukturen, Unions und typedef-Deklarationen:

Datenorganisation in C-Programmen

Sicherlich haben die meisten von Ihnen beim Programmieren in C schon Strukturen, Unions und typedef-Deklarationen benutzt. Im allgemeinen sind Strukturen einfach zu handhaben und unterstützen geradezu beispielhaft den korrekten Umgang mit Daten. Ferner dienen sie dazu, den Code lesbarer und wartungsfreundlicher zu gestalten. Mit Unions und typedef-Deklarationen kann man dies auch erreichen, doch ist ihr Gebrauch schwieriger und wird leicht unübersichtlich.

Weil der Gebrauch der Unions und Strukturen schon ziemlich weit verbreitet ist, will ich hier auf die grundlegende Syntax (die Regeln, mit denen sie innerhalb der Sprache C aufgebaut werden) und Semantik (ihre Bedeutung) nicht näher eingehen. Statt dessen sollen hier einige typische Eigenarten, die beim Gebrauch auftreten, beschrieben werden. Dem Leser soll damit geholfen werden, Programmierfehler zu vermeiden, Probleme der Übertragbarkeit (Portabilität) von Programmen zu erkennen und überhaupt die Sprache C ein wenig besser verstehen.

Neue Entwicklungen

Der ursprüngliche Sprachumfang von C wurde in dem Klassiker »The C Programming Language« von Brian W. Kernighan und Dennis M. Ritchie (im folgenden: K&R) beschrieben. In dieser Sprachdefinition waren Wertzuweisungen an Strukturen, Strukturen als Argumente und Funktionen, die Strukturen zum Ergebnis hatten, nicht erlaubt. Heutige Compiler hingegen lassen solche Anweisungen zu. Diese Sprachelemente wurden in den Normungsvorschlag des »American National Standard Institue for C« (ANSI C) aufgenommen. Dieser Vorschlag soll, soweit ich weiß, nicht mehr überarbeitet werden und dürfte bald als allgemeiner Standard anerkannt sein.

Ein anderes neues Charakteristikum, das in dem Normungsvorschlag aufgenommen wurde, ist das Initialisieren automatischer Strukturen. Allerdings ist es nicht in allen Compilerversionen verfügbar, besonders in denen nicht, die üblicherweise im Xenix- und Unix-Compiler Entwicklungspaket enthalten sind. Zum Beispiel würden solche Compiler die Struktur-Deklaration in *Listing 1A* nicht akzeptieren. Hier muß man zusätzlich eine static-Vereinbarung hinzufügen (siehe *Listing 1B*).

Dies ist für den Programmierer umständlich; der Grund liegt eigentlich nur darin, daß eine ältere Sprachversion zugrunde gelegt wird und ist nicht etwa durch Zwänge, die sich aus der Compilerkonstruktion ergeben, begründet. Diese Einschränkung kann Sie als Programmierer aber auch dazu zwingen, mehr statischen Speicherplatz zu belegen, als Ihnen lieb ist. Wenn es auf Speicherplatz ankommt, kann das schon problematisch werden. Als Alternative bietet sich eigentlich nur das explizite Zuweisen von Werten an die einzelnen Elemente der Struktur im Code. In Bezug auf Geschwindigkeit ist das auch unerheblich, denn beim Erstellen des lauffähigen Programms macht der Compiler nichts anderes, jedoch wird Ihr Programm dadurch nicht gerade lesbarer. Bei automatischen Unions und arrays tritt dieses Problem ebenfalls auf. Sie sind zwar möglich, jedoch von ANSI nicht vorgeschrieben. Da sie aber allgemein üblich sind und durch die

A

```
1      main()
2      {
3           struct {
4                int a;
5           } s = { 1 };
6      }
```

B

```
1      main()
2      {
3           static struct {
4                int a;
5           } s = { 1 };
6      }
```

C

```
1      main()
2      {
3           struct {
4                int a;
5           } s1, s2;
6
7           if (s1 == s2)        /* syntaktischer Fehler */
8                printf("s1 == s2\n");
9           else
10               printf("s1 != s2\n");
11     }
```

◄◄ *Listing 1: Einfache Beispiele für Struktur-Deklarationen*

Normierung unterstützt werden, würde ich mir den Kauf von Compilern, die sie nicht ermöglichen, zweimal überlegen.

Der Vergleich von Strukturen

Versuchen Sie das Programmbeispiel aus *Listing 1C* zu kompilieren. Wie Sie sehen, meldet der Compiler einen syntaktischen Fehler. Dieser Fehler entsteht dadurch, daß man Strukturen und auch Unions nicht als Ganzes auf Gleichheit überprüfen kann (im Gegensatz zur Wertzuweisung struct1 = struct2, die erlaubt ist). Die einzige Möglichkeit zwei Strukturen zu vergleichen, ist der Abgleich jeder einzelnen Komponente.

Das hört sich erstmal nach einer unnötigen Einschränkung beim Gebrauch von Strukturen an (so wie die umständliche Initialisierung). Aber es gibt sogar zwei Gründe dafür: Zum einen wäre zu erwarten, daß außer Vergleichen auch Prüfungen auf Ungleichheit (!=), relationale Operatoren (>, <, <=, >=) und vielleicht noch ganz andere Dinge erlaubt sind. Und das ist nicht unbedingt wünschenswert, weil dadurch der Compileraufbau sehr kompliziert würde. Darüber hinaus kann es durchaus vorkommen, daß eine Struktur extrem viel Speicherplatz belegt (darauf gehe ich nachher noch genauer ein) und die Erzeugung von fehlerfreiem Code mehr als umständlich wird (ganz besonders bei den relationalen Operatoren). Außerdem wird es außerordentlich schwierig, bei Cross-Kompilierung, genauer der Cross-Assemblierung, bestimmte Fehler aufzuspüren.

Zugriff auf Struktur-Elemente

In C gibt es zwei Operatoren, die den Zugriff auf die einzelnen Elemente ermöglichen, und den meisten Programmierern wird ihr Gebrauch auch nicht besonders schwer fallen. Es sind . (Punkt) und -> (Pfeil). Den Punkt benutzt man, um direkt auf ein Element zuzugreifen (bei Strukturen und Unions). Erfolgt der Zugriff über einen Zeiger, der auf die Struktur oder Union weist, wird der Pfeil benutzt. Im Zusammenhang mit anderen Operatoren jedoch scheint manchmal einige Verwirrung zu herrschen. Die größten Probleme tauchen offenbar beim Gebrauch von & (Adresse von) und * (Verweisoperator) auf. Für Anfänger sind Konstellationen wie &*p-> ebenso verständlich wie Hieroglyphen. In *Listing 2* sind einige Beispiele aufgeführt, und nicht nur Anfänger werden bei einigen Anweisungen ihre Schwierigkeiten haben, genau sagen zu können, was gemeint ist.

```
1        struct s {
2             int y;
3             int *x;
4        };
5
6       struct s foo;
7       struct s *pfoo = &foo;
8
9       main()
10      {
11           printf("%d\n, foo.x);
12           printf("%d\n, pfoo->x);
13           printf("%d\n, &pfoo->x);
14           printf("%d\n, *pfoo->x);
15           printf("%d\n, &foo.x);
16           printf("%d\n, *&pfoo->x);
17           printf("%d\n, &*pfoo->x);
18           printf("%d\n, sizeof(int));
19           printf("%d\n, &foo);
20       }
```

◄ *Listing 2: Selektion von Struktur-Komponenten*

Der Gebrauch der einzelnen Operatoren zum Zugriff auf Struktur-Komponenten richtet sich ganz besonders nach der Priorität der Operatoren. In der Liste der Prioritäten (mehr darüber finden Sie in der Sprachbeschreibung des Microsoft C Compilers 5.1, den ich im folgenden MSC nenne) stehen Punkt und Pfeil ganz oben (sie werden also zusammen mit den Klammern () und [] vorrangig verarbeitet). Daran sollte man stets denken. Die Priorität ist auch, wie im »Komplexe C-Deklarationen verständlich gemacht«, Micorsoft System Journal (Jg.3, Heft 1) beschrieben, ein wichtiger Hinweis zum Verständnis von C-Anweisungen. Überhaupt ist die Kenntnis der verschiedenen Prioritäten eine grundlegende Voraussetzung für C-Programmierer. Es ist natürlich wesentlich einfacher, Programme zu lesen, zu schreiben und zu pflegen, wenn man sich dieser Prioritäten stets bewußt ist und nicht ständig nachschlagen muß. Das kann gar nicht oft genug betont werden!

Listing 2 dürfte nun verständlicher sein. Das Problem liegt wahrscheinlich in Zeile 13. Verfolgt man diese Anweisung Schritt für Schritt, so sollte man zum Ergebnis kommen, wie es im *Listing 3* angegeben ist. In Zeile 12 wird die Adresse des Elements x ermittelt, auf dessen Struktur pfoo indirekt zeigt. Allgemein weist pfoo->x auf das Element x einer Struktur. Es wird also nicht die Adresse von pfoo als Zeiger auf die Struktur mit dem Element x benutzt.

▸ *Listing 3: Festsetzung des Vorrangs von Operatoren*

```
1       &(pfoo->x)
2       *(pfoo->x)
3       &(foo.x)
4       &(*(pfoo->x))  oder einfach  pfoo->x
5       *(&(pfoo->x))  oder einfach  pfoo->x
```

In Zeile 14 wird der Wert von x ermittelt. Dabei ist x Komponente der Struktur, auf die pfoo zeigt. Das ist deshalb möglich, da x als int*-Variable deklariert wurde. Allerdings wird nicht der Wert der Variablen pfoo als struct-Zeiger benutzt, um x zu referieren. Ansonsten müßte es das gleiche sein wie pfoo->x, was ganz offensichtlich nicht funktioniert.

Zeile 16 ist besonders interessant, weil sich einige der Operatoren gegenseitig aufheben. Angenommen, x ist als int deklariert. Nun können wir *&x codieren. Zuerst wird die Adresse von x berechnet und in einer int* Variable abgelegt. Danach wird der durch diesen Wert adressierte Inhalt des Speichers ermittelt. Natürlich hätte man statt dessen auch einfach auf x zugreifen können. So etwas ähnliches passiert in Zeile 16. Es wird der Inhalt der Adresse von pfoo->x bestimmt. (Beachten Sie hierbei wieder die Prioritäten der Operatoren!)

Zum Schluß betrachten wir Zeile 17. Obwohl sie sich etwas von Zeile 16 unterscheidet, ist das Ergebnis identisch, wenn auch etwas schwieriger zu verstehen. Sehen wir uns also *Listing 4* an. Man sollte erwarten, daß &*pvar in Zeile 10 das gleiche bedeutet wie &9999, weil *pvar dem Wert 9999 entspricht (natürlich ergibt das so keinen Sinn, weil man schlecht die Adresse einer Konstanten angeben kann). Wenn wir also *pvar als Adresse des Inhalts von pvar interpretieren, ist das Ergebnis &var, da var der Inhalt von pvar ist. Mit dieser Argumentationsweise wird auch Zeile 17 in Listing 2 erklärt.

▸ *Listing 4: Weitergehende Beispiele zur Selektion von Struktur-Komponenten*

```
1       main()
2       {
3           int       var = 9999;
4           int       *pvar = &var;
5
6           printf("%d\n, pvar);
7           printf("%d\n, *pvar);
8           printf("%d\n, &pvar);
9           printf("%d\n, *&pvar);
10          printf("%d\n, &*pvar);
11          printf("\n);
12          printf("%d\n, var);
13          printf("%d\n, &var);
14          printf("%d\n, *&var);
15          printf("%d\n, &*&var);
16          printf("%d\n, *&*&var);
17      }
```

Die Zeilen 15, 18 und 19 dienen dem Selbststudium. Wie würden die Format-Anweisungen aussehen, wenn x als *char definiert wäre?

Die Speicherorganisation von Strukturen

Eine andere unangenehme und etwas unverständliche Erscheinung beim Umgang mit Strukturen ist ihre Größe. Grundsätzlich ist das, was man sieht, nicht unbedingt identisch mit dem, was tatsächlich vorhanden ist. Aber wieso sollte das den Anwender interessieren?

Ich möchte das Problem am nächsten Programmbeispiel (siehe *Listing 5*) näher erläutern. Wenn das Programm auf einem 80386-Rechner abläuft, erhält man die angegebenen Ergebnisse. Sizeof(char) ergibt 1 und sizeof(int) ergibt 4 als Byte- bzw. Wortlänge auf einem 80386-Rechner (unter Unix und beim Large-Modell unter DOS). Bei der Angabe der Größe von chara und charb geht auch alles glatt, nur scheint bei huh irgend etwas nicht zu stimmen. Es handelt sich jedoch weder um einen Programmierfehler, noch um eine Macke im Compiler. Wir sind hier auf ein Phänomen gestoßen, das in C zu Recht als undefiniert gilt. In unserem Beispiel belegt huh acht Bytes statt der erwarteten fünf, weil intb vier Bytes belegt und auf einer Wort-Adresse, die durch vier teilbar ist, liegen muß. Der Compiler schiebt deshalb nach dem einem Byte, das chara belegt, drei undefinierte Bytes ein, damit die intb auch wirklich an der richtigen Stelle aufhört.

▸▸ *Listing 5: Programmbeispiel für Größen von Strukturen*

```
1       #define offsetof(type, identifier) (&(((type *)0)-\
        >identifier))
2
3       main()
4       {
5           struct chara {
6               char      chara;
7           };
8
9           struct charab {
10              char    chara;
11              char    charb;
12          };
13
14          struct huh {
15              char    chara;
16              int     intb;
17          };
18
19          struct huh2 {
20              int     intb;
21              char    chara;
22          };
23
24          printf("%d\n, sizeof(char));
25          printf("%d\n, sizeof(int));
26
27          printf("%d\n, sizeof(struct chara));
28          printf("%d\n, sizeof(struct charab));
29          printf("%d\n, sizeof(struct huh));
30          printf("%d\n, sizeof(struct huh2));
31
32          printf("%d\n, offsetof(struct charab, charb));
33          printf("%d\n, offsetof(struct huh, intb));
34          printf("%d\n, offsetof(struct huh2, intb));
35      }

Die Ausgabe dieses Programms auf einem 80386-Rechner lautet:

1
4
1
2
8
8
1
4
0
```

Dadurch, daß man die Komponenten einfach umdreht, wird der Speicherplatz allerdings auch nicht besser ausgenutzt. Das wird deutlich, wenn man die Größe von huh2 betrachtet. Liegt hier nun eine Schwäche des Compilers vor, der nicht optimiert arbeitet, oder gibt es dafür einen vernünftigen Grund? Stellt man sich ein Array vom

Typ huh2 vor, dann sieht man, daß diese Art der Speicherbelegung notwendig ist, damit alle Feldelemente auf einer Wort-Adresse beginnen.

Geht man einen Schritt weiter, so bedeutet das, daß auch jede Struktur im Speicher so ausgerichtet ist, eine Tatsache, die auf den ersten Blick wohl nicht so offensichtlich ist. Durch diese Art der Ausrichtung wird allerdings zur Ausführungszeit kein Speicher belegt, sie ist nur zur Anpassung an die Systemumgebung notwendig, tritt also so nicht auf allen CPUs auf. Bei einigen Systemen werden Variablen nur auf geradzahligen Adressen abgelegt, bei anderen auf Vielfache von 2, 4, 8, ... Bytes. Wieder andere erlauben zwar, die Art der Ausrichtung zu wählen, und bieten damit eine nicht so strenge aber dafür auch nicht so effiziente Ausrichtung. Microsoft C (MSC) Version 5.1 bietet diese Option, die Sie aktivieren können, wenn der Speicher zu stark fragmentiert ist und dadurch Speicherplatz knapp wird. Sie kann durch Setzen des /ZP-Schalters in der Kommandozeile oder durch die #pragma pack-Anweisung im Programmcode ausgewählt werden. Zu beiden Alternativen finden Sie genaueres im Benutzerhandbuch (Kapitel 3.3.15) des Microsoft C Optimizing Compiler Version 5.1.

Das Lesen und Schreiben von Strukturen

Es ist häufig notwendig, eine Struktur in einer Diskettendatei oder im RAM abzulegen (oder einem Kanal oder einer anderen Interprozess-Kommunikation in Betriebssystemen wie OS/2, Xenix oder Unix). Da ein Programm, das auf diese Struktur zugreift, nicht notwendigerweise mit den gleichen Compileroptionen erstellt wurde oder sogar auf einer anderen Maschine läuft als jenes, das Daten in die Struktur hineinschreibt, ist es oft sinnvoll, mit einem kleinen Programm die Werte der Strukturen auszugeben (einen Dump erzeugen). Man kann nicht einfach davon ausgehen, daß die Daten richtig sind. Denn, wie oben dargestellt, entstehen durch die Angleichung der Adressen Löcher, die die physikalische Anordnung der Struktur verändern. Eigentlich kann man von gar nichts ausgehen, ohne sich den Quellcode und die Compileroptionen genau anzusehen.

Die Bestimmung des Offsets

Hardwareabhängige Konstanten im Programm zu verwenden, wird allgemein als schlechter Programmierstil angesehen. Wegen der unter Umständen auftretenden Ausrichtung der Strukturen im Speicher ist es weder schön noch portabel, den Offset einer Komponente in einer Struktur durch hardwareabhängige Werte darzustellen. Kehren wir noch einmal zum Listing 5 zurück. Wie wir gesehen haben, ist es nicht ganz einfach zu bestimmen, in welchem Byte intb beginnt. Statt es direkt anzugeben, sollten Sie lieber das offsetof-Makro verwenden.

Das offsetof-Makro steht bei den meisten neueren Compilern zur Verfügung und befindet sich in der Datei <stddef.h>. An dieses Makro werden als Parameter die Struktur und die Komponente übergeben. Als Ergebnis wird der Offset dieses Elements, gerechnet vom Anfang der Struktur, in Byte zurückgegeben. Listing 5 enthält eine mögliche Implementation dieses Makros (#define offsetof in Zeile 1), die auf den meisten Rechnern funktionieren sollte. Die grundsätzliche Idee hinter diesem Makro ist die Annahme, daß die Struktur ab Adresse 0 beginnt [folglich (type*)0], und somit eine Referenz auf eine beliebige Komponente den relativen Offset in Byte ergibt. Zum Experimentieren mit Strukturen können Sie die Anweisungen aus Listing 5 benutzen.

Typenanpassung

Bis jetzt haben wir uns hauptsächlich mit Strukturen beschäftigt. Ein weiteres Sprachelement von C sind Unions, die vom syntaktischen Aufbau Strukturen sehr ähnlich, in der Bedeutung jedoch völlig anders sind. Leider wird ihre Wirkungsweise in C-Programmen oft falsch verstanden.

Um das zu erläutern, werde ich erst einmal darlegen, was Unions sind und was sie nicht sind (vielleicht besser nicht zu sein scheinen?). Eine Union ist eine Variable, die zu einem bestimmten Zeitpunkt ein (und genau ein) Objekt vieler verschiedener benannter Objekttypen beinhalten kann (also Objekte, deren Speicherplätze sich überlappen), unabhängig vom Typ dieser Objekte. Daraus lassen sich folgende Schlüsse ziehen:

- Zweck von Unions ist es, Speicherplatz oder Variablen mehrfach zu verwenden.
- Alle Komponenten einer Union sind an derselben Startadresse im Speicher abgelegt.
- Die Größe einer Union wird durch die Größe ihrer größten Komponente festgelegt.
- Nur ein einziger Komponentenwert kann in einer Union abgelegt sein.
- Aus Gründen des Programmierstils sollten Sie sicherstellen, daß alle Komponenten der Union einer speziellen logischen Einheit des Programms zugeordnet ist, in der sie benutzt werden.

Damit stellen Unions ein nützliches Hilfsmittel zur Typangleichung und zur Neudefinition von Typen dar. Auf die Neudefinition gehe ich im nächsten Abschnitt ein, zunächst soll es um die Typangleichung gehen.

Bei der Systemprogrammierung ist es oft notwendig, bei der Benutzung von Systemresourcen, Einheitentreibern oder bestimmter Hardware

(z.B. DMA) Datenformate anzupassen. C als die De-facto-Standardsprache zur Systemprogrammierung stellt dafür Unions zur Verfügung.

Wir können zum Beispiel davon ausgehen, daß die meisten Betriebssysteme oder Einheiten auf Wortgrenzen arbeiten (was, um genau zu sein, bei den meisten Maschinen eine Begrenzung auf geraden Adressen bedeutet). Weiterhin können wir mit einiger Sicherheit annehmen, daß eine Hardwareeinheit Informationen mit einer Länge von 6 Byte erwartet. Wenn wir nun einfach eine Variable als char info[6] deklarieren, so ist es keineswegs sicher, daß der C-Compiler diesen Wert auf einer geraden Speicheradresse ablegt. Damit haben wir gerade eine 50 prozentige Chance – und noch ungleich geringere, den Fehler zu finden, wenn 6 Monate später das Programm geändert wird.

Um das Problem zu lösen, also die Variable sicher auf einer geraden Speicheradresse abzulegen, deklariert man sie in der folgenden Weise:

```
union device_data {
int   dummy;   /* ausrichten */
char  info[6];
};
```

Da dummy eine gerade Adresse besitzt, und alle Komponenten einer Union auf derselben Startadresse beginnen, liegt info auch auf einer Wortgrenze. Natürlich brauchen Sie sich in Ihrem Programm um den Wert dummy nicht mehr zu kümmern, er hat seinen Zweck erfüllt und wir das Problem doch recht elegant gelöst, oder?

Redefinitionen – Zweckentfremdung von Unions?

Da C nicht vorschreibt, wie Unions verwendet werden sollen, besonders die Benutzung von nur einer Komponente zu einer bestimmten Zeit keine zwingende Einschränkung der Sprache ist, bleibt dem Programmierer überlassen, wie er damit arbeitet. Mit anderen Worten: Sie können die Struktur einer Union einsetzen, wie es am zweckmäßigsten ist. Sie müssen nur daran denken, daß die einem Wert zugeordnete Komponente nur solange verfügbar ist, bis eine andere Komponente der Struktur zugewiesen wird. Das Problem besteht eben darin, daß in C dafür keinerlei Kontrollen vorgesehen sind und sich der Programmierer innerhalb der syntaktischen Granzen frei entfalten kann.

Ist zum Beispiel deklariert:

```
union example {
  int     i ;
  double  d ;
  } ex ;
int    j;
```

so wird bei den folgenden Anweisungen kein syntaktischer Fehler auftreten:

```
ex.d = 999.999;
j    = i;
```

Allerdings können Sie davon ausgehen, daß j keinen vernünftigen Wert enthält, wenn Sie nicht gerade eine Zufallszahl erzeugen wollten. Trotz allem ist es bei einiger Umsicht durchaus möglich, diese Nebeneffekte sinnvoll zu nutzen. Dann sollte man aber zum einen völlig verstanden haben, was passiert, und außerdem sauber und ausführlich dokumentieren!

Im letzten Jahr habe ich zum Beispiel an einem Auftragsprojekt gearbeitet, bei dem das Betriebssystem RMX von Intel eingesetzt wurde. Es lief auf einem 80386-Rechner, also einer Maschine mit einer segmentierten Architektur. Irgendwann benötigten wir den Zugriff auf dynamischen Speicherplatz. Aus einem ziemlich kuriosen Grund hat es sich gezeigt, daß keine der Standardroutinen von C (z.B. malloc) zur Problemlösung geeignet war. Statt dessen verwendeten wir eine RMX-Systemroutine, um den Speicher zu erhalten. Daraus ergab sich das Problem, daß wir als Ergebnis dieser Routine ein sogenanntes Token erhielten, das sich bei jedem Routinenaufruf als Startadresse eines Segments herausstellte. Das Token enthielt also jedesmal die zwei Byte der Segmentadresse, nicht jedoch den Offset innerhalb des Segments (ebenfalls zwei Byte), dabei benötigten wir dieses Token als Zeiger auf einen Character. Unsere Lösung war ein kleines Programm, das so ähnlich aussah wie in *Listing 6A*. Allerdings ist es aus den oben genannten Gründen keineswegs sicher, daß es mit allen Compilern läuft. Außerdem gingen wir davon aus, daß pointer == int zulässig ist. Darüber hinaus ist die Anordnung von Offset- und Segmentadresse innerhalb der Variablen systemabhängig, da die interne Darstellung von Worten und Bytes von der Hardware abhängt. Daran sieht man recht deutlich, daß die ganze Lösung nicht besonders sauber ist.

▸▸ *Listing 6A: Beispiel für einen nicht standardgemäßen Gebrauch von Unions*

```
1       #define getmemory()     20 /* simulierter RSX-Aufruf */
2
3       char    *
4       toktopr(token)
5       short  token;
6       {
7           union {
8               struct {
9                   short offset;
10                  short segment;
11              } segoff;
12              char    *ptr;
13          } convert;
14
15      convert.segoff.segment = token;  /* hole Segmentnummer */
16      convert.segoff.offset = 0;  /* Offset im Segment ist Null */
17
18      return (convert.ptr); /* exakte Übernahme des Zeigers
                                 auf Segment und Offset */
19      }
20
21      main()
22      {
23          int token = getmemory();  /* RSX Speicheraufruf */
24          char   *p = toktoptr(token);
25
26          printf("%d\n", p);
27      }
```

Listing 6B: Programmbeispiel aus 6a in Standard-C umgeschrieben

```
1       #define getmemory()     20 /* simulierter RSX-Aufruf */
2
3       char    *
4       toktopr(token)
5       short  token;
6       {
7          return ((char *) ((long(token) << (sizeof(short) * 8)));
8       }
9
10      main()
11      {
12          int token = getmemory();  /* RSX Speicheraufruf */
13          char    *p = toktoptr(token);
14
15          printf("%d\n", p);
16      }
```

Trotz allem ist eine derart einfache Umwandlungsroutine erstaunlich funktionsfähig. Bevor wir uns aber zu sehr darüber freuen, sollten wir uns die bevorzugte Methode der Typangleichung ansehen, die cast-Operation. *Listing 6B* ist eine Version der toktoptr-Umwandlung, die mit cast-Operationen arbeitet. Diese Lösung sieht zwar nicht gerade besonders schön oder elegant aus und ist auch durch ein paar getroffene Annahmen nicht ohne weiteres portabel, zeigt aber sehr deutlich, daß Unions zur Redefinition benutzt werden können, dieses aber einige schwere Nachteile mit sich bringt. Wenn Unions nicht zu ihrem eigentlichen Zweck als mehrfach nutzbare Speicher verwendet werden, ist es ganz einfach nicht zulässig und vollkommen nichtportabel, sogar zwischen verschiedenen Compilern auf demselben Rechner. Es ist darüber hinaus nicht einmal auszuschließen, daß optimierende Compiler inkorrekten Code aus den nicht zulässigen Anweisungen erzeugen. Die Verwendung von cast-Operationen ist die, obwohl auch systemabhängige, im allgemeinen weniger problematische Methode und außerdem »legales« C.

Der Grund, diese Nebeneffekte so ausführlich darzustellen ist nicht, Ihnen neue Programmiertricks aufzuzeigen, sondern Sie auf schlechten Programmierstil hinzuweisen und Sie vorzubereiten, falls Ihnen derartiger Code beim Bearbeiten oder Pflegen eines Programms vorkommen sollte. Wenn Sie sich doch entscheiden, etwas Ähnliches selbst zu codieren, kennen Sie jetzt die Probleme und was passieren kann.

Der allgemeine Gebrauch von Unions

Eine weitere Anwendung von Unions ist das Erzeugen von codierten Records. Es ist zulässig, in einer Union einen gültigen Variablennamen zu deklarieren, und somit innerhalb einer Union eine Struktur zu verwenden (und umgekehrt). *Listing 7* enthält ein Beispiel zur Verarbeitung eines bestimmten Records, der etwa über eine Kommandozeile o.ä., eingegeben wurde. Jeder Record besitzt eine Kennung int recordtype, mit der, abhängig vom Aussehen des Records, der Programmablauf über die switch-Anweisung gesteuert werden kann. Im zugehörigen Programmteil werden dann die entsprechenden Komponenten des Records bearbeitet. Damit bietet C eine sehr elegante und wartungsfreundliche Methode, um Datenstrukturen verarbeiten zu können, die ohne unnötigen Speicherplatz oder Zeiger auf komplizierte Datenstrukturen auskommen. Insbesondere ist es möglich, im Gegensatz zum oben Besprochenen, den ersten int-Wert der Struktur coderecord ganz problemlos wiederzuverwenden. Darüber hinaus bieten sich überlappendende Strukturen noch einen weiteren Vorteil: Sind bei sich überlappenden Strukturen innerhalb einer Union die Anfangskomponenten gleich, kann man z.B. die eine Komponente beschreiben und den Wert aus der anderen lesen. Wenn etwa die Strukturen type1 und type2 identisch aufgebaut sind, ist es durchaus zulässig, einen beliebigen Wert in a zu speichern und danach c zu benutzen, ohne daß irgendwelche Nebeneffekte auftreten können. Die Komponenten a und c sind vom gleichen Datentyp und liegen auf demselben Offset innerhalb der Union. Vom Programmierstil ist es übrigens sauberer, die Komponenten a, c und e aus der Union herauszulösen und als einzige Komponente der Struktur genericrecord zu verwenden. Allerdings hängt dieses ganz wesentlich vom logischen Zusammenhang zwischen den Bezeichnern im Programmcode ab.

Listing 7: Verbundene unions und structs

```
1       #include <stdio.h>
2
3       union codedrecords {
4           struct
5               int     a;
6               float   b;
7           } typel;
8           struct {
9               int     c;
10              long    d;
11          } type2;
12          struct {
13              int     e;
14              int     f;
15          } type3;
16      };
17
18      struct genericrecord {
19          int     recordtype;
20          union   codedrecords cr;
21      };
22
23      struct genericrecord gr;
24
25      main()
26      {
27          /* ... */
28
29          switch(genericrecord.recordtype) {
30          case TYPE1:
31              /* Anweisungen mit genericrecord.cr.typel.? */
32          break;
33
34          case TYPE2:
35              /* Anweisungen mit genericrecord.cr.type2.? */
36          break;
37
38          case TYPE3:
39              /* Anweisungen mit genericrecord.cr.type3.? */
40          break;
41
42          default:
43              fprintf(stderr, "Falscher Recordtyp!!!");
44          exit (1);
45              break;
46          }
47
48          /* ... */
49      }
```

Noch eine letzte Anmerkung zu Unions: obwohl in der Vorschlagsliste des ANSI C die Initia-

lisierung von Unions erlaubt ist, sind hier zwei Einschränkungen zu beachten. Zum einen darf nur durch Konstanten initialisiert werden, die zweitens nur die erste Komponente festlegt. Darauf sollten Sie ganz besonders achten, wenn Ihr Compiler den Typ konvertiert und die Zuweisung z.B. durch Abschneiden der Nachkommastellen vornimmt. Unter Umständen reicht es dann aus, die Reihenfolge der Komponenten zu verändern, um eine korrekte Initialisierung zu gewährleisten.

Die typedef-Deklaration

Bevor ich auf typedef-Deklarationen und Strukturen eingehe, möchte ich in diesem Zusammenhang noch auf einige Mißverständnisse hinweisen. Zum einen ist das reservierte Wort typedef zwar syntaktisch als Speicherzuweisung definiert, reserviert aber zur Laufzeit keinen Platz. Damit ist die Bezeichnung eigentlich falsch, dient aber einer konsistenten und geeigneten Notation.

Zweitens und für unsere Betrachtung wichtiger ist, daß durch eine typedef-Deklaration kein neuer Variablentyp definiert wird, auch wenn dies der Name nahelegt. Mit einer typedef-Deklaration kann nur ein Name für einen der Grundtypen oder einer Untermenge davon neu vergeben werden. Darüber hinaus wird dieser neue Name noch in eine Tabelle eingetragen, die die einzelnen Bezeichner ordnet, sie befindet sich in der Symboltabelle des Compilers (general name space). Diese Tabelle umfaßt außerdem die Funktionen, die meisten Variablen und Aufzählungstypen. Damit unterscheiden sich die durch typedef definierten Namen nicht von anderen Variablennamen und dienen lediglich zur Klassifizierung von Bezeichnern, die durch sie festgelegt werden.

Letztlich verwirrt, daß es nicht zulässig ist, den Bezeichner einer Speicherklasse im Typnamen der typedef-Deklaration zu benutzen. Der Grund dafür liegt darin, daß innerhalb einer Deklaration keine zwei Bezeichner von Speicherklassen verwendet werden dürfen. Man könnte argumentieren, daß dadurch schwer lesbare Programme vermieden werden, da Speicherattribute nicht in der typedef-Deklaration versteckt wären, also ein sauberer Programmierstil erzwungen wird. Mit anderen Worten: die Attribute einer Variablen, die aufgrund einer typedef-Deklaration definiert werden, könnte man sonst nicht klar genug erkennen. Es ist allerdings fraglich, ob ein objektorientierter Programmierer dieser Argumentation zustimmen kann.

Speicherbelegung von Strukturen

Es gibt zwei Möglichkeiten, Speicherplatz für Strukturen zu reservieren, durch #define und typedef. Mit beiden Deklarationen können ähnliche Ergebnisse erzielt werden, sie unterscheiden sich allerdings durch syntaktischen Aufbau und ganz besonders in ihrer Bedeutung. Zur Speicherbelegung sollte im allgemeinen die typedef-Deklaration verwenden. Und zwar aus folgendem Grund:

Da K&R sich nicht besonders ausführlich über den Gebrauch von typedef-Deklarationen ausgelassen haben und die früheren C-Compiler (außer von AT&T) diese oftmals nicht unterstützt haben (vermutlich aus demselben Grund), ignorierten viele Programmierer bis vor ein paar Jahren typedef-Deklarationen oder benutzten sie falsch. Die gängigste Methode der Deklaration war die #define-Direktive. Bevor das reservierte Wort void in allen Compilern implementiert war, wurde z.B.

```
#define void int;
```

in das Programm importiert (meist über ein eigene Include-Datei wie mydefs.h), ähnlich den gebräuchlichen Makros der Datei stdio.h. Obwohl das im allgemeinen durchaus funktioniert (kompilieren Sie doch einfach den Code in *Listing 8*), treten schon bei etwas komplizierteren Definitionen Probleme auf, und zwar deshalb, weil #define nur als reine Textersetzung zu verstehen ist, die durch den Präprozessor erledigt wird. Nur kümmert der sich nicht im geringsten um die Syntax von C. Er ersetzt lediglich einen Text durch einen anderen. Solange man ein fehlerfreies C-Programm in den Präprozessor hineingibt, wird auch ein fehlerfreies Programm herauskommen. Auf der anderen Seite bewertet der Compiler typedef- Namen und kann innerhalb von Ausdrücken auf Konsistenz der typedef-Deklarationen prüfen.

▸▸ *Listing 8: Vergleiche zwischen #define und typedef*

A
```
1       #define v int      /* Da void ein Schlüsselwort ist, wird es
                                              hier nicht benutzt */
2
3       main()
4       {
5           int x;
6           v   y;
7
8           x = y;
9       }
```

B
```
1       type int v;        /* Da void ein Schlüsselwort ist, wird es
                                              hier nicht benutzt */
2
3       main()
4       {
5           int x;
6           v   y;
7
8           x = y;
9       }
```

Wenn wir uns die wenigen, von K&R gelieferten Beispiele ansehen, wird deutlich, warum die #define-Direktive nicht funktioniert: Zum Beispiel die Deklaration

```
typedef char * String;
```

entspricht folgendem #define-Statement:

```
#define MAXLINES (5)
#define String char *
```

Ein Beispiel zur Stringverarbeitung:

```
String p,
lineptr [MAXLINES],
alloc(int);
```

Auf den ersten Blick existieren keine Probleme. Folgen wir allerdings der Textersetzung dieser Zeilen in das, was der C-Compiler erhält, so ergibt sich mit:

```
char * p, lineptr[5],
alloc(int);
```

sicherlich nicht das beabsichtigte Ergebnis.

Es wurde nur p als char * deklariert, lineptr als ein Array von char und alloc als eine Funktion mit einen char-Wert als Ergebnis. Beabsichtigt war aber:

```
char * p, *lineptr[5],
*alloc(int);
```

Neben dem Problem mit der #define-Direktive wird hier übrigens die Gefahr deutlich, in einer Deklaration mehrere Deklarationstypen zu verwenden. Das kann sich als wahrer Fallstrick erweisen! Mehr darüber finden Sie in meinem oben erwähnten Artikel.

Ein anderes Beispiel dafür, daß #define nicht immer fuktioniert, ist

```
typedef int* array 20[20];
```

Hier gibt es einfach keine Möglichkeit, die Deklaration

```
array20  samplearray ;
```

zu erzeugen.

Zum Schluß sollten wir noch, um den Bogen zu Strukturen und Unions zu schließen, auf die Typenprüfung des Compilers eingehen. Sehen wir uns das Beispiel Treenode aus K&R an, lautet die entsprechende Definition (von Treeptr in diesem Zusammenhang einmal abgesehen):

```
#define Treenode struct {\
char   *word;\
int     count;\
}
```

die in der folgenden Form in einem Programm verwendet werden könnte:

```
Treenode   tn1, tn2;
Treenode   tn3, *ptn;
```

Trotzdem tn1 und tn2 vom gleichen Typ und vollständig kompatibel sind, haben diese Variablen nichts mit den anderen, die an anderer Stelle mit Treenode definiert werden, gemeinsam. Auch dann, wenn es sich, wie ptn, um Zeiger handelt. Aus diesem Grund sind die Anweisungen

```
tn1 = tn2;
tn1.count = tn2.count;
```

korrekt, aber die Anweisungen

```
tn1 = tn3;
ptn = &tn1;
```

führen zweifellos auf syntaktische Fehler. Um es noch deutlicher zu machen, verwenden wir einmal für jede Definition eine eigene Anweisung:

```
Treenode tn1 ;
Treenode tn2 ;
Treenode tn3 ;
Treenode *ptn;
```

Damit sind die vier Variablen nicht im geringsten miteinander zu kombinieren. Ersetzen wir nämlich Treenode durch die entsprechende #define-Direktive, wie es der Präprozessor tut, so erhalten wir vier voneinander unabhängige, unbenannte struct-Deklarationen. Sie sind zwar alle zufällig gleich aufgebaut, aber den Compiler interessiert das natürlich wenig.

Wird statt dessen die typedef-Deklaration

```
typedef struct {
char  * word;
int   count;
} Treenode;
```

vorgenommen, sind alle oben aufgeführten Zuweisungen zulässig. In der Symboltabelle des Compilers wird nun ein einziger Eintrag für Treenode aufgenommen und die anschließende Typenprüfung führt nicht mehr auf einen syntaktischen Fehler. Da die typedef-Deklaration echten Code darstellt, sind alle durch Treenode definierten Variablen vom gleichen Typ.

Aussichten

Strukturen und Unions sind klar definiert, und die meisten Programmierer verwenden sie auch einigermaßen erfolgreich. Um sie jedoch in ihrem vollen Umfang und vor allem portabel nutzen zu können, muß man sich allerdings etwas genauer als allgemein üblich mit den Feinheiten auskennen. Das trifft auch für die typedef-Deklaration zu, besonders weil sie oft falsch oder überhaupt nicht benutzt wird (Typedef-Deklarationen werden unter OS/2 und in der Umgebung des Presentation Managers besonders wichtig). Strukturen sind die wichtigsten Hilfsmittel zur Konstruktion komplexer Daten in C (und einigen anderen Sprachen), und nun, da Sie einiges Neues darüber erfahren haben, können Sie die Sprache C ein wenig besser verstehen. Insbesondere für einen vernünftigen Programmierstil und der Entwicklung von portablen Programmen sind diese Sprachelemente enorm wichtig.

Details einer Programmiersprache nicht zu kennen ist nie besonders sinnvoll. Es ist zwar sicherlich einiges überflüssig oder schlecht umgesetzt, aber ganze Teile des Sprachumfangs von

C nicht zu nutzen, nur weil sie ein bißchen anspruchsvoller sind, sollte einem Programmierer nicht in den Sinn kommen. Trotzdem C keine einfache Sprache ist und zuweilen regelrecht kryptisch codiert, führt das Verständnis der feineren und komplizierteren Strukturen zur Entfaltung der vollen Mächtigkeit von C.

Nach meinen eigenen Erfahrungen mit C (und denen vieler anderer) bringt es wenig, in C zu programmieren, wenn man die Sprache nicht wirklich verstanden hat. Üblicherweise verlangsamt das die Programmentwicklung und verursacht diverse Fehler, und das kostet Geld. Wenn Sie aber am Ball bleiben und sich die Mühe machen, die weiterführende Syntax und die dahinterstehende Philosophie zu verstehen, so wird Ihnen das Beste, was C zu bieten hat, zur Verfügung stehen.

Greg Comeau

Greg Comeau ist einer der Gründer von Comeau Computing, einer unabhängigen Softwarefirma, die sich besonders auf UNIX- und C-Entwicklungswerkzeuge spezialisiert hat. Außerdem berät und schult er UNIX- und C-Benutzer.

RAPID Prototyping:

Hilfe für C-Entwickler

Den Begriff des Rapid Prototyping kennt man häufig von RDBMS-Entwicklungsumgebungen (RDBMS = Relationales Datenbank-Management-System) unter DOS und UNIX (z.B. bei Informix, RBase) bzw. von deren Sprachen der 4. Generation. Solche Sprachen ermöglichen eine nichtprozedurale Programmierung. Für den Entwickler stellt sich also nicht die Frage »Wie löse ich mein Problem«, sondern »Was ist mein Problem«. Die Lösungsstrategien beinhaltet die Sprache selbst, die durch ihre Spezialisierung auf bestimmte Anwendungsgebiete über mächtige, vorgefertigte Funktionen verfügen.

Diese Sprachen haben einen bestimmten Fokus und sind damit auf spezielle Anwendungsgebiete beschränkt (z.B. auf Datenbank-Anwendungen im kaufmännischen Bereich). Mit ihrer Hilfe lassen sich für solche Anwendungen schnell Dialogoberflächen aufbauen, Aktionen aus fertigen Modulbibliotheken anbinden, Datenbank-Schemata definieren und modifizieren. Der Entwickler kann seinem Auftraggeber schnell einen Prototypen anbieten und anschließend zum fertigen Anwendungssystem weiterentwickeln.

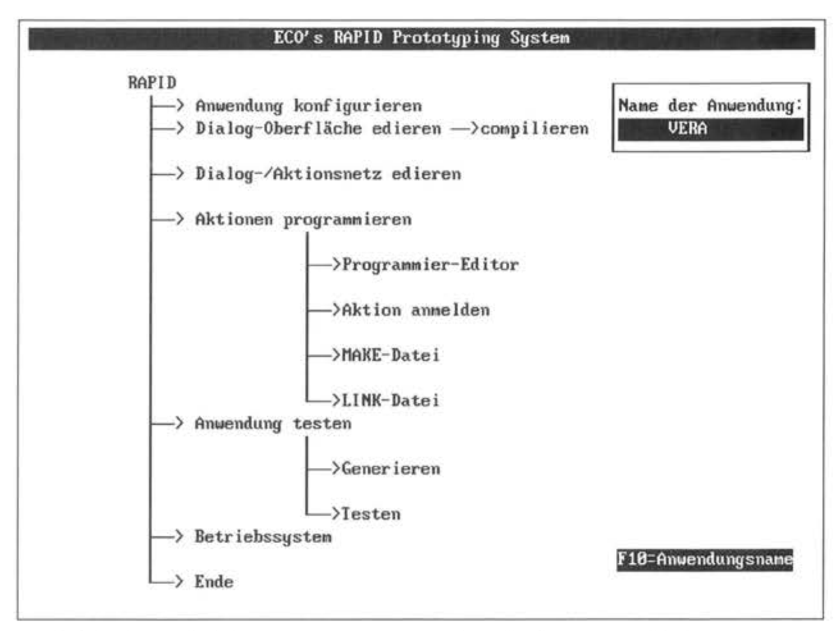


Bild 1: Abbildung des gesamten Entwicklungszyklus einer Applikation.

Solche Entwicklungsumgebungen sind äußerst nützlich, aber für viele Anwendungsgebiete nicht geeignet. Wer mit Sprachen der 3. Generation (General Purpose-Sprachen) entwickeln muß (z.B. für die Entwicklung technischer Anwendungen aus dem CAD/CAM-Bereich findet kaum Unterstützung durch flexible, leistungsfähige RAPID Prototyping-Systeme. Sofern sie überhaupt angeboten werden, kann damit zwar ein Prototyp erstellt werden – die eigentliche Applikation muß dann jedoch in einer geeigneten Programmiersprache neu geschrieben werden (z.B. bei Dan Bricklin's DEMO II). Anders verhält es sich bei dem RAPID Prototyping-System, das vom ECO Institut entwickelt wurde.

Das Gesamtpaket des RAPID Prototyping-Systems beinhaltet als Hauptkomponenten ein User Interface-Toolkit für die Gestaltung von Dialogoberflächen sowie ein User Interface Management-System für die Steuerung der Übergänge zwischen Dialog- und Aktionszuständen.

Die Komponenten des RAPID Prototyping-Systems

Die Oberfläche von »ECO's RAPID Prototyping-System« bildet den gesamten Entwicklungszyklus einer Applikation ab (*Bild 1*). RAPID erzwingt ein systematisches Vorgehen bei der Entwicklung: Es

muß zunächst ein Systementwurf vorliegen, und die Applikation muß etwa in einem Dialog- und Aktionsgraphen beschrieben sein.

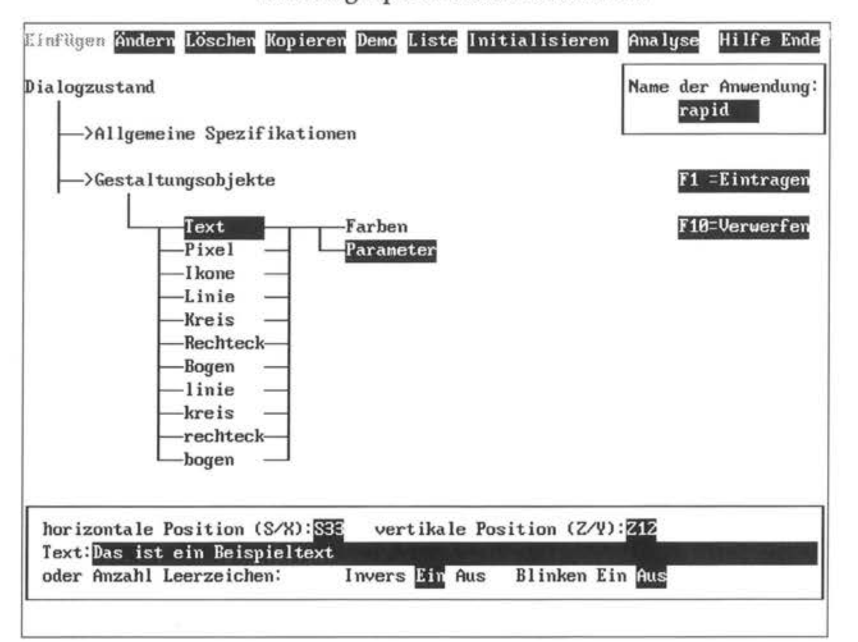


Bild 2: Unmittelbarer Test und Modifikation von Dialogzuständen.

Anschließend werden die Dialogzustände gestaltet. Das geschieht objektorientiert mit Hilfe des speziellen Editors DIALEDIT, d.h. der Designer der Benutzeroberfläche muß keine bestimmte Spezifikationssprache erlernen. Dialogzustände können unmittelbar getestet und beliebig modifiziert werden (*Bild 2*). DIALEDIT erstellt und ändert automatisch eine Resource-Datei, <anwendung>.IPR, die die Beschreibung sämtlicher Dialogzustände einer Applikation enthält.

Löschen Einfügen Ersetzen Kopieren Verschieben Suchen Hilfe Ende

Zustand		Deskriptor	D11	D11	D11	D11	D11	D11	D11
Aktion		Maskenfeld	2	3	4	5	6	7	8
		Regeln							
<-	->	Obligat.							
<=	=>	Datentyp	S1	S2	S2	S2	S2	S2	S2
<==	==>	Vorzeichen							
Up	Dn	Wert							
Pup	Pdn	Default	Aktion	Neu	Ändern	Löschen	Einfüge	Ersetze	Kopiere
Anf	End	Kontext							
		Rel. Zustand							
		Rel. Feld							
		Rel. Wert							
		Relation							
		Akt. Wert							
		Fehl.Zustand							
		Verknüpfung							
		Aut.Übergang							
		Bedingung							
		Zielzustand							
		Zielfeld							
		Fortsetzung							

Bild 3: Das Dialogverhalten kann ohne Kenntnis einer Spezifikationssprache beschrieben werden.

Im nächsten Schritt müssen die Wege zwischen den Zuständen einer Applikation beschrieben werden. Zulässige Wege aus einem bzw. in einen Zustand sind von Übergangsbedingungen abhängig, also von der Interpretation der Benutzerwillensäußerungen oder Aktions-Rückgabewerten und ggf. von dem Ergebnis kontextfreier oder kontextsensitiver Prüfungen von Benutzereingaben. Ebenfalls mit Hilfe eines speziellen Werkzeuges, dem Editor RWTEDIT, kann dieses Regelwerk schrittweise aufgebaut und immer mehr verfeinert werden. Auch bei diesem Schritt kann der Designer der Benutzerschnittstelle ohne jegliche Kenntnis einer Spezifikationssprache das komplette Dialogverhalten einer Applikation beschreiben (*Bild 3*). Die entsprechenden Ressource-Dateien dieser Regel- und Wegetabellen (RWT) werden von RWTEdit automatisch erstellt und modifiziert.

Die Dialog- und Aktionsmaschine AUTODIAL ist ein vollständig generischer Zustandsautomat, der sein Wissen über eine Applikation aus den Recource-Dateien erhält. Damit kann ein Prototyp einer Applikation schnell und ohne Programmierkenntnisse aufgebaut und beliebig geändert werden, ohne daß hierfür eine Zeile Quellcode produziert werden muß.

Schließlich müssen die Aktionen einer Applikation programmiert werden, sofern sie anwendungsspezifisch sind und noch nicht als Modul zur Verfügung stehen. Die Aktionen werden sukzessiv als Zustände in die Regel- und Wegetabelle integriert und nach dem Zusammenbinden mit der Applikation über AUTODIAL zum entsprechenden Zeitpunkt bzw. unter bestimmten Bedingungen gestartet, die in der Regel- und Wegetabelle definiert sind. Den Zusammenhang der verschiedenen Komponenten vom RAPID Prototyping-System zeigt *Bild 4*.

Der Prototyp einer Applikation kann ohne großen Aufwand solange modifiziert oder ergänzt werden, bis das gewünschte Applikationsdesign erreicht ist. Die Applikation steht damit auch gleichzeitig in endgültiger Form für den Realeinsatz zur Verfügung und muß nicht mehr – wie bei früheren Prototyping Systemen – in einer geeigneten Programmiersprache neu programmiert werden.

Bisherige Erfahrungen

Derzeit befassen sich verschiedene Firmen z.B. auch die Grundig AG oder die Happy User Software GmbH mit dem Einsatz des neuen RAPID Prototyping-Systems. Die Grundig-Evaluierungsstudie, die gegenwärtig noch bearbeitet wird, soll feststellen, ob mit dem Prototyping-System eine schnelle Realisierung von Benutzerschnittstellen auch unter XENIX möglich ist und ob die Dialog- und Aktionsmaschine die Interaktionsverwaltung zwischen dem Benutzer und den Modulen leisten kann. Man hofft, bei der Entwicklung mit RAPID Zeit und Kosten einsparen zu können, indem Fehler und Mängel, die bei Verwendung eines Pflichtenheftes als wesentliche Entwicklungsgrundlage erst relativ spät festgestellt werden können, durch Einsatz eines Prototyping-Systems bereits frühzeitig erkannt werden.

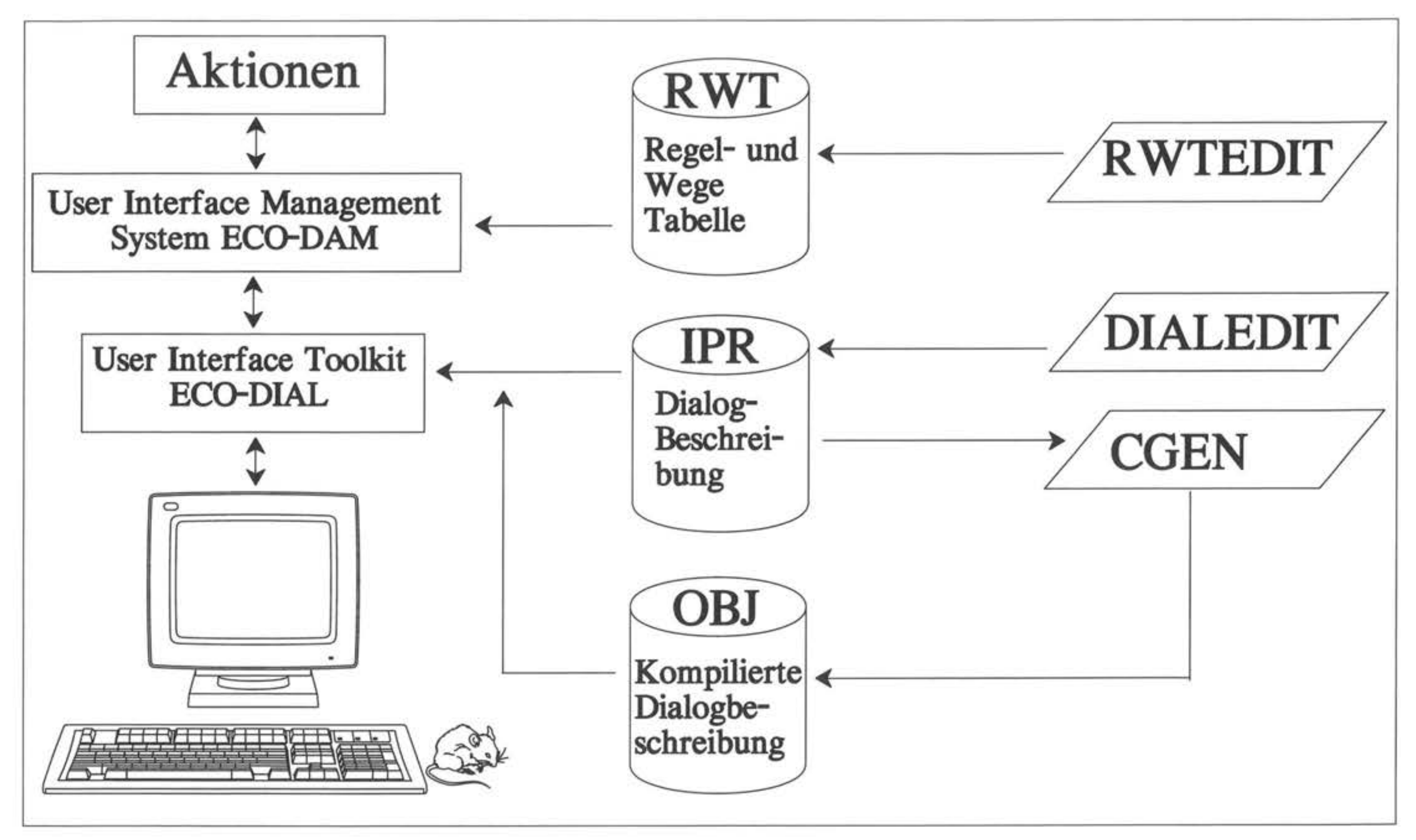


Bild 4: Die verschiedenen Komponenten des RAPID Prototyping-Systems.

Vor allem könnten auch verschiedene Varianten simuliert und daraus wichtige Gestaltungshinweise für die Entwicklung gewonnen werden. Nicht zuletzt bliebe diese Flexibilität als Resultat aus dem Einsatz der Dialog- und Aktionsmaschine auch im realisierten Endprodukt erhalten.

Wünsche und Anforderungen von Endanwendern ließen sich so relativ einfach und schnell berücksichtigen.

Stand der Entwicklung

Das Gesamtpaket RAPID wird inzwischen für MS-DOS im Text- und Grapikmodus und für XENIX im Textmodus angeboten. Der Grafikmodus unter XENIX auf der Basis von X-Windows soll bis Ende 1989 realisiert sein.

Rainer von Ammon, ECO Institut, Landshuter Str. 37, 8400 Regensburg 1, Tel.: (0941) 74833

Neue Telefonnummern für die Microsoft Hotline

Mit dem Umzug der deutschen Microsoft Niederlassung nach Unterschleißheim bei München konnten die Hotline-Kapazitäten verdreifacht werden. Erstmals steht auch für den Macintosh-Bereich ein eigener Telefonanschluß zur Verfügung. Auch die Öffnungszeiten der Hotline wurden verlängert. Die Microsoft Mitarbeiter stehen Montag bis Freitag zwischen 9 und 12 Uhr sowie zwischen 13 und 16 Uhr für telefonische Produktinformationen zur Verfügung.

Mehr als 5000 Anrufe werden jeden Monat von den Microsoft Support-Mitarbeitern entgegengenommen. Hinzu kommen monatlich rund 2000 schriftliche Anfragen zu Betriebssystemen, Standardapplikationen und Sprachen. Mit der jetzt erreichten Erweiterung der Telefonanlage will Microsoft künftig noch mehr Anrufe bewältigen, Wartezeiten verkürzen und damit ihren Support für ihre Kunden verbessern.

Für die verschiedenen Produkte stehen folgende Durchwahlnummern zur Verfügung:

Telefonnr.	Produkte
089/31705-81	Microsoft Produkte für Macintosh
089/31705-82	Microsoft Multiplan, Microsoft Excel, Microsoft Chart, Microsoft Works
089/31705-83	Microsoft Word, Microsoft Windows Write, Word Druckertreiber
089/31705-84	Microsoft Flightsimulator, Microsoft Windows Utilities, Microsoft Paintbrush, EasyCad
089/31705-85	Microsoft Windows/286, Microsoft Windows/386, Microsoft Project, Microsoft Maus, Netzwerkapplikationen, Extended/Expanded Memory, LIM Standard
089/31705-86	Microsoft Compiler, Microsoft Quick Sprachen

Microsoft C-Kurs für Umsteiger, Teil 1:

Programmieren in C: Die Grundlagen

Ab dieser Ausgabe des *Microsoft System Journals* bringen wir in mehreren Folgen einen C-Kurs für Umsteiger, als für Leute, die bereits Erfahrungen mit anderen Programmiersprachen wie Pascal oder Basic gemacht haben. Der Kurs wird die gesamte Palette der C-Programmierung zum Inhalt haben, von den Grundlagen unter QuickC bis zur Anwendungsprogrammierung unter Windows.

0. Einleitung

Die Programmiersprache C hat sich in den letzten Jahren zu der am häufigsten verwendeten Sprache in der professionellen Software-Entwicklung herausgebildet. Dies ist im wesentlichen auf die effiziente Code-Generierung der C-Compiler, den standardisierten Sprachumfang und die flexiblen Einsatzgebiete von der System- bis zur Applikations-Entwicklung zurückzuführen. Außerdem gibt es im Umfeld von C eine Reihe von leistungsfähigen Werkzeugen und Toolboxen, die die Produktivität der Entwickler steigern. Für die Betriebssysteme MS-DOS, OS/2 und SCO XENIX / SCO UNIX ist Microsoft der führende Anbieter von C-Compilern. Alle C-Compiler von Microsoft unterstützen den vollen C-Sprachumfang mit einer Reihe von sogenannten ANSI-Erweiterungen und erzeugen einen optimierten Code. Außerdem beinhalten die Produkte eine umfangreiche Runtime-Library, einen sehr komfortablen Debugger, mit dem Programm-Tracing auf C-Sourcecode-Level möglich ist, und eine ganze Reihe von anderen nützlichen Utilities. Dieser C-Kurs soll Ihnen die Programmiersprache C näher bringen. Deshalb haben wir auch viele Beispiele eingearbeitet. Die Beispiele können Sie sofort mit Hilfe von Quick-C ausprobieren. Wir verwenden die Version 2.0 in englisch; Ihre Befehlsfolgen können von den hier angegebenen abweichen, wenn Sie mit der deutschen Version arbeiten.

0.1 Die 0 als Mittelpunkt der Programmiersprache

Ein wesentlicher Punkt von C ist, daß alles mit 0 beginnt oder endet:

- in C wird immer von 0 gezählt
- C benutzt die 0 für falsch und ungleich 0 für wahr
- Feldgrenzen beginnen bei 0
- Strings hören mit dem 0-Zeichen auf
- C-Zeiger benutzen 0 als Wert für einen nicht belegten Zeiger

C kennt nur Funktionen und wenige sogenannte Schlüsselwörter. Es existieren keine Sprachelemente, um Informationen auf dem Bildschirm oder der Tastatur aus- bzw. einzugeben (analog INPUT und PRINT in Basic). Dies wird nur mit Hilfe von Funktionen realisiert. Auch Stringvergleiche sind als Funktionen implementiert.

0.2 Warum C ?

C ist eine kleine Sprache. Sie ist leicht zu erlernen (wenige Schlüsselwörter) und gut auf den jeweiligen Prozessor umzusetzen (kompakter Code). Durch den Einsatz der richtigen Kontrollstrukturen und Datentypen können nahezu alle gestellten Probleme gelöst werden. Durch den Mechanismus, alles in Funktionen auszulagern, konnte

der Sprachumfang sehr früh in dem Buch »The Programming Language« /1/ von Kerninghan & Ritchie dokumentiert werden und wurde so quasi normiert.

C ist portabel. Durch den kleinen Sprachumfang können nicht nur C-Programme, sondern auch der C-Compiler selbst schnell portiert werden. Deshalb werden auch als erstes C-Compiler bei neuen Betriebssystemen angeboten (OS/2). Die Runtime-Library des Compilers ist ebenfalls in C geschrieben.

C ist prägnant. Durch eine Vielzahl von Operatoren kann der Sourcecode kompakt und knapp formuliert werden. Ein einziges C-Statement kann in anderen Programmiersprachen unter Umständen nur in mehreren Anweisungen programmiert werden. Des weiteren sind alle Operatoren auch auf Zeigern definiert, so daß auch mit Zeigern gerechnet werden kann.

C ist modular. Ein C-Programm besteht aus einer beliebigen Anzahl von externen Funktionen, die innerhalb des gesamten Programms aufrufbar sind. C kennt also keine eingebetteten Funktionen die Bestanddteil einer anderen Funktion sind. Jedes C-Programm enthält genau eine Main-Funktion (main()). Diese wird nach dem Programmstart als erste aufgerufen. Wenn die Main-Funktion terminiert, ist das C-Programm beendet.

C ist großzügig. C bietet viele Möglichkeiten, die bei einem Mißbrauch zu unübersichtlicher Programmstruktur und Laufzeitfehlern führen können. Es wird deshalb eine gewisse Disziplin vom Entwickler verlangt. So ist C vollkommen formatfrei. Eine Anweisung kann eine beliebige Anzahl von Blanks, Tabs oder Newlines enthalten. Die Vielzahl von Operatoren kann bei übertriebener Anwendung zu einer unverständlichen Programmstruktur führen. Wir empfehlen besonders dem C-Neuling eine defensive Programmierung. Nicht alles, was in C möglich ist, läßt sich mit den Grundsätzen einer klaren und strukturierten Softwareentwickluung vereinbaren.

0.3 Aufbau des Seminars

Das Seminar behandelt detailliert den Aufbau der Sprache C. Es werden sämtliche Sprachelemente vorgestellt und anhand zahlreicher Beispiele erläutert. Die Beispiele sind mit dem MSC 5.1 und dem Quick-C-Compiler 2.0 entwickelt und getestet. Jeder Seminarteil ist als eigenständige Einheit konzipiert und unterteilt sich in folgende Kategorien:

Einleitung: Das Thema des aktuellen Seminarteils wird vorgestellt.

Sprachelemente: Sprachelemente werden vorgestellt und anhand von Beispielen erläutert. Die Beispiele sind als Sourcecodefragmente und als ablauffähige Programme und Funktionen aufgebaut. Sie werden mit Hilfe des Quick C-Compilers Version 2.0 im Editor erfaßt und ausprobiert.

Library Guide: Die Beispielprogramme benutzen Funktionen der Standardbibliothek. Diese werden im Rahmen des Library Guide aufgegriffen und erläutert.

Quick-C und MS C-Compiler: Es wird auf Compileroptionen und Besonderheiten der beiden Compiler hingewiesen.

Utilities: Der MS C-Compiler verfügt über zahlreiche Dienstprogramme, die im Laufe der Serie vorgestellt werden.

Aufgaben: Am Ende eines jeden Seminars werden Aufgaben gestellt, die die Möglichkeit bieten, das Gelernte zu festigen und zu vertiefen. Dabei wird nur auf das vermittelte Wissen der jeweiligen Folge zurückgegriffen. Musterlösungen der Aufgaben werden in der nächsten Folge veröffentlicht.

Zusammenfassung und Ausblick: Jeder Seminarteil wird durch eine Zusammenfassung abgeschlossen. Es wird ein Ausblick auf die nächste Folge des C-Kurses gegeben.

Der erste Seminarteil befaßt sich mit den Grundkonstrukten der Sprache. Dieses Seminar geht davon aus, daß der Leser über Grundkenntnisse der Programmierung verfügt und in anderen Hochsprachen (Fortran, Basic, Pascal etc.) schon Software erstellt hat. C steht in dem Ruf ebenso schwierig und fehleranfällig wie effizient und portabel zu sein. Wir werden sehen, daß C vielleicht ein wenig mehr Disziplin erfordert als andere Hochsprachen, aber auch enorme Möglichkeiten bietet. Jedenfalls ist C alles andere als schwierig und im Rahmen dieses Kurses problemlos zu erlernen.

```
main ()
{
    printf ("\nDas ist mein erstes C-Programm\n");
    printf ("Es besteht nur aus der main-Funktion\n");
    printf ("Die Funktion liefert als Funktionswert 0\n");
    printf ("an die rufende Funktion\n");
}
```

◂◂ *Abb. 1: Das erste Programm*

0.4 Das erste C-Programm

Wir werden das erste C-Programm im Editor der Quick-C-Entwicklungsumgebung erfassen. Dazu rufen Sie Quick-C im Betriebssystem folgendermaßen auf:

```
qc
```

Geben Sie das in *Abb. 1* dargestellte Programm ein. Anschließend sichern Sie das Programm unter dem Namen ansi.c, indem Sie die Tasten Alt-F drücken. Es klappt dann auf der linken Bildschirmseite das File-Menu herunter. Sie können dann mit Hilfe der Cursortasten und der Return-Taste oder durch Eingabe eines 'a' (= Save As) das Dateisicherungsfenster öffnen. Geben Sie dann als Dateinamen ansi.c ein und drücken Sie die Return-Taste. Damit ist Ihr erstes Programm unter dem Namen ansi.c abgespeichert.

Das Sourcefile ansi.c enhält die Funktion main(). Diese Funktion wird beim Programm-

start als erstes aufgerufen. Sie muß in jedem ausführbaren C-Programm vorhanden sein. Jede Funktion besteht aus einem Funktionskopf und einem Funktionsrumpf.

Der Funktionskopf enthält die Namen der Parameter der Funktion in runden Klammern (). Nach der Aufzählung der Variablen werden die Datentypen festgelegt. In unserem Fall werden keine Parameter übergeben.

Der Funktionsrumpf enthält die Variablendefinition und die Anweisungen der Funktion. Der Funktionsrumpf wird immer durch { } eingerahmt. Am Anfang der Funktionen stehen die Variablendeklarationen gefolgt von den Anweisungen. Funktionswerte werden an die rufende Funktion mit Hilfe des Schlüsselwortes return übergeben. Wenn kein Funktionsdatentyp angegeben wurde, wird int angenommen.

In unserem Beispiel erwartet die Funktion main() keine Parameter und liefert auch kein Ergebnis. Der Funktionstyp ist void.

0.5 Übersetzen des ersten Programms

Bitte übersetzen Sie jetzt das Programm und führen Sie es aus. In Quick-C geschieht dies durch Drücken der Taste F5. Innerhalb eines Windows wird Ihnen dann mitgeteilt, wieviel Zeilen übersetzt, wieviele Fehler und Warnungen entdeckt wurden. Anschließend wird der Linker aufgerufen, der aus dem Objektmodul ein ausführbares Programm erzeugt. Nachdem das Programm erzeugt wurde, wird es sofort ausgeführt. Sie erhalten als Ausgabe auf dem Bildschirm:

```
Das ist mein erstes C-Programm
Es besteht nur aus der main-Funktion
Die Funktion liefert als Funktionswert 0
an die rufende Funktion
```

1. Sprachelemente

Wir werden uns in dieser Folge mit den grundlegenden Sprachelementen beschäftigen. Nach den Datentypen, Variablen, Konstanten und Zuweisungen wenden wir uns dem Präprozessor und den Ein-/Ausgabefunktionen zu. Anschließend werden die Operatoren und die Kontrollstrukturen behandelt.

1.1 Datentypen, Variablen, Konstanten und Zuweisungen

Eine Variablendeklaration in C besteht aus der Speicherklasse, dem Datentyp und dem Variablennamen. Die Deklaration

```
int i;
```

definiert eine Variable vom Datentyp int (Integer).

Variablennamen können bis zu 31 Zeichen lang sein und aus Buchstaben, Ziffern und Underscore (_) bestehen. Ziffern dürfen jedoch nicht am Anfang stehen. C ist »case sensitiv«, d.h. es unterscheidet Groß- und Kleinschreibung. Die Bezeichner i und I kennzeichnen also zwei verschiedene Variablen.

Lebensdauer und Gültigkeit von Variablen

C ist eine blockorientierte Programmiersprache. Das bedeutet, daß alle zusammenhängenden Anweisungen in Blöcken gruppiert werden. Blöcke werden durch geschweifte Klammern {} eingeschlossen. Im ersten Programm wurden alle Anweisungen der Funktion main in einem Block (dem Funktionsrumpf) zusammengefaßt.

Variablen, die innerhalb von Blöcken deklariert werden, sind nur solange gültig wie auch der Block selbst gültig ist. Andere Funktionen können also nicht auf die lokalen (=innerhalb eines Blocks definierten) Variablen einer Funktion zugreifen. Neben diesen lokalen Variablen gibt es noch globale Variablen, die außerhalb einer Funktion definiert werden und von allen Funktionen benutzt werden können.

Die elementaren Datentypen

char: Ist ein Byte groß und dient zur Speicherung ganzzahliger Werte. Der Datentyp char eignet sich besonders, um den ASCII-Wert eines Zeichens abzuspeichern (Wertebereich: -127 bis 127).

int: Ist ein maschinenabhängiger Datentyp und dient zur Speicherung ganzer Zahlen. Ein int ist in Abhängigkeit vom verwendeten Prozessor (16-Bit 2 Byte, 32-Bit 4 Byte) 2 oder 4 Byte groß (Wertebereich 2-Byte-Integer: -32767 bis 32767, 4-Byte-Integer: -2147483647 bis 2147483647).

short int: Ist ein kurzer Integer, er ist immer 2 Byte groß. Statt short int schreibt man kürzer short (Wertebereich -32767 bis 32767).

long int: Ist ein langer Integer, immer 4 Byte. Statt long int schreibt man kürzer long (Wertebereich -2147483647 bis 2147483647).

float: Gleitkommazahl mit einfacher Genauigkeit. Ist 4 Byte groß.

double: Gleitkommazahl mit doppelter Genauigkeit. Die Größe beträgt 8 Byte.

Alle ganzzahligen Datentypen (char, int, short, long) können mit dem Attribut unsigned versehen werden. Ihr Wertebereich ist dann positiv.

char	0	...	255
short	0	...	65.535
long	0	...	4.294.967.395

Der Datentyp char ist maschinenunabhängig. Seine Größe beträgt immer 1 Byte. Der Datentyp int ist maschinenabhängig. Selbst die Datentypen short oder long sind von der Größe nicht genormt. Es ist lediglich festgelegt, daß ein short nicht größer sein darf als ein int und ein int nicht größer als ein long. Bei allen Compilern, die den Autoren bekannt sind, ist ein short 2 Byte und ein long 4 Byte lang.

Komplexe Datentypen

Aus allen elementaren Datentypen können komplexere Datentypen (Arrays oder Strukturen) gebildet werden. So wird ein Feld eines elemen-

taren Datentyps definiert, indem an den Variablennamen die Dimension des Feldes in eckigen Klammern vermerkt wird:

```
char str[10];
int  ifeld[5];
```

deklariert Felder mit 10 bzw. 5 Elementen. Wie eingangs bereits erwähnt, fängt in C alles mit der 0 an. So ist das erste Element str[0] bzw. ifeld[0]. Das letzte Element die str[9] bzw. ifeld[4]. Die C-Besonderheit wird von vielen C-Anfängern oft vergessen. Da C Feldgrenzen beim Zugriff nicht überprüft, können sehr leicht Programmfehler entstehen.

Strukturen werden in einer späteren Folge detailliert besprochen.

Nun aber wieder ran an die Maschine. Das nächste Programm wartet. Geben Sie das Programm der *Abb. 2* ein, übersetzen Sie es und führen Sie es aus. In dem Programm werden Variablen aller elementaren Datentypen deklariert. Der sizeof-Operator wird benutzt, um die Größe einer Variablen oder eines Variablentyps in Byte zu ermitteln. Bitte beachten Sie, daß nicht eine zusammenhängende Zeichenkette angegeben wurde, sondern mehrere einzelne. Diese werden beim Übersetzen zu einer zusammengefügt. Wichtig ist, daß die einzelnen Strings nur durch sogenannte »white space characters« (Blanks, Tabs, Newlines) getrennt sein dürfen.

```
main ()
{
    char        c;
    short       s;
    int         i;
    long        l;
    float       f;
    double      d;
    long double ld;

    printf ("\n\tGröße elementarer Datentypen in Byte:\n\n"
            "\tchar\t\t%d\n"
            "\tshort\t\t%d\n"
            "\tint\t\t%d\n"
            "\tlong\t\t%d\n"
            "\tfloat\t\t%d\n"
            "\tdouble\t\t%d\n"
            "\tlong double\t%d\n\n",
            sizeof(c), sizeof(s), sizeof(i),
            sizeof (l), sizeof(f), sizeof (d), sizeof (ld));
}
```

◀◀ *Abb. 2: Größen einzelner Datentypen*

Konstanten

Die Zuordnung einer Konstanten zu einem Datentyp wird über die Deklaration einer Konstanten realisiert. C kennt folgende Konstanten:

Ganzahlige numerische Konstanten können im Dezimal-, Oktal- oder Hexadezimalsystem angegeben werden. Oktale Konstanten beginnen mit einer 0 (z.B. 0177 entspricht 255), Konstanten im Hexadezimalsystem beginnen mit 0x (z.B. 0xff entspricht 255).

Endet die Konstante mit l bzw. L, so wird eine long int-Konstante deklariert. Konstanten mit dem Suffix U bzw. u haben den Datentyp unsigned int.

Gebrochene numerische Konstanten sind immer vom Datentyp double und werden mit dem Dezimalpunkt und/oder in E-Notation angegeben. Beispiele: 12.34, 1.234E1, 1234e-2

Character-Konstanten werden in einfachen Hochkommata eingeschlossen. 'A' steht für den ASCII-Wert 65 (A). Alle Zeichen lassen sich über einen Backslash gefolgt von einer oktalen Konstanten darstellen:

```
'A' oder '\081'
```

bezeichnen beide den ASCII-Wert des Zeichens 'A'. Die Sprache C definiert eine Reihe von Character-Konstanten als Escape-Sequenzen:

\a	Alert,Piepton
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage Return
\t	Horizontaler Tabsprung
\v	Vertikaler Tabsprung
\\	Backslash
\?	Fragezeichen
\'	Einfaches Hochkomma
\"	Doppeltes Hochkomma
\nnn	Oktalwert (\081)
\xnn	Hexadezimalwert (\x1b).

Stringkonstanten werden in doppelte Anführungszeichen eingeschlossen ("Dies ist eine Zeichenkette"). Stringkonstanten werden im Datensegment abgelegt und mit einem Nullbyte ('\0') versehen. Das Nullbyte markiert in C das Ende eines Strings. Stringkonstanten können durch den Compiler verkettet werden. Insbesondere bei langen mehrzeiligen Zeichenketten erhöht diese ANSI-Erweiterung die Lesbarkeit eines Programms (s.a. Abb. 2).

Aufzählungskonstanten (enumerated type) sind symbolische Namen, denen ganzzahlige Werte zugeordnet werden. Die symbolischen Namen erhöhen die Lesbarkeit eines Programms. Es ist allerdings nicht definiert, daß Compiler die Gültigkeit einer symbolischen Zuweisung überprüfen.

Ändern Sie das Programm aus Abb. 2 so ab, daß Sie den einzelnen Variablen nach der Deklaration Konstanten zuweisen und innerhalb des Aufrufs der Funktion printf alle sizeof-Operatoren entfernen. Die Zuweisung in C erfolgt mit dem Operator =. Beispiel:

```
int i;

i= 1234;
```

1.2 Der Präprozessor

Der Präprozessor ist ein Textprozessor, der als erstes bei einer Compilation aufgerufen wird. Er führt Präprozessordirektiven aus, die alle mit einem »#« eingeleitet werden. Die wichtigsten Direktiven (#define und #include) werden typischerweise dazu verwendet, Programme übersichtlicher und portabler zu machen, indem maschinenabhängige Programmteile in Headerfiles ausgelagert und Konstanten durch symboli-

sche Namen ersetzt werden. Wir werden die beiden Direktiven #define und #include innerhalb dieses Abschnitts anhand von Beispielen erläutern.

Die Präprozessordirektive #define

Wir beginnen hier mit einem durchgängigen Beispiel, um alle folgende Sprachelemente zu beschreiben. Das Programm wird von Sprachelement zu Sprachelement erweitert oder verändert, damit Sie die Anwendung der Sprachelemente direkt anhand eines Programms ausprobieren können.

Wir wollen eine kleine Bildschirmsteuerung mit Hilfe der ANSI-Escapesequenzen programmieren. Sehen Sie sich bitte das Programm der *Abb. 3* an und geben Sie es ein. Sie sehen, daß alle Stringkonstanten mit Hilfe der #define-Anweisung oberhalb der Funktion main deklariert werden. Der Präprozessor ersetzt alle in der Funktion enthaltenen Textkonstanten durch die nach der Konstanten angegebenen Strings.

► *Abb. 3: Programm mit ANSI-Escapesequenzen*

```
#define ANSI_CLR      "\033[2J"
#define ANSI_INVERS   "\033[7m"
#define ANSI_NORMAL   "\033[0m"
#define ANSI_CURPOS   "\033[%u;%uH"
#define TEXT          "Dieser inverse Text beginnt an Zeile/Spalte %d/%d"

main ()
{
    printf (ANSI_CLR);
    printf (ANSI_INVERS);
    printf (ANSI_CURPOS, 5, 10);
    printf (TEXT, 5, 10);
    printf (ANSI_NORMAL);
    printf (ANSI_CURPOS, 23, 1);
}
```

Übersetzen Sie das Programm und bringen Sie es zur Ausführung.

Neben diesem reinen Textersatz, ist die Verwendung von Parametern in #define-Anweisungen besonders interessant. Dieses sogenannte Makro wird genauso aufgerufen wie eine Funktion. Der Präprozessor ersetzt das Makro mit seinen Platzhaltern. Dies ist insbesondere dann sinnvoll, wenn eine Funktion nur eine Anweisung hat, da so der durch einen zusätzlichen Funktionsaufruf verursachte Laufzeitoverhead (Parameterübergabe, Anlegen der lokalen Variablen) vermieden wird.

Ändern Sie das Programm der Abb. 3 so ab, daß die Cursorposition an das Makro ANSI_CURPOS übergeben wird. Dazu müssen Sie die printf-Funktion ebenfalls in die #define-Anweisung integrieren. In *Abb. 4a* wurden alle Makros mit Parametern versehen.

► *Abb. 4a: ansi.c*

```
#define ANSI_INVERS       7
#define ANSI_NORMAL       0
#define ANSI_CLR()        printf ("\033[2J")
#define ANSI_ATTR(a)      printf ("\033[%um",a)
#define ANSI_CURPOS(z,s) printf ("\033[%u;%uH",z,s)
#define TEXT(z,s)         printf ("Dieser Text beginnt an Zeile/Spalte
%d/%d",z,s)

main ()
{
    /* Bildschirm löschen */
    ANSI_CLR();

    /* Attribut invers anschalten */
    ANSI_ATTR(ANSI_INVERS);

    /* Cursor an Position zeile = 5 und spalte = 10 setzen */
    ANSI_CURPOS(5,10);

    /* Text ausgeben an zeile = 5 und spalte = 10 */
    TEXT(5,10);

    /* Cursor an Position zeile = 7 und spalte = 10 setzen */
    ANSI_CURPOS(7,10);

    /* Attribut normal anschalten */
    ANSI_ATTR(ANSI_NORMAL);
}
```

►► *Abb. 4b: ckurs.h*

```
#define ANSI_INVERS       7
#define ANSI_NORMAL       0
#define ANSI_CLR()        printf ("\033[2J")
#define ANSI_ATTR(a)      printf ("\033[%um",a)
#define ANSI_CURPOS(z,s) printf ("\033[%u;%uH",z,s)
#define TEXT(z,s)         printf ("Dieser Text beginnt an Zeile/Spalte
%d/%d",z,s)
```

►► *Abb. 4c: ansi1.c*

```
#include  "ckurs1.h"

main ()
{
    /* Bildschirm löschen */
    ANSI_CLR();

    /* Attribut invers anschalten */
    ANSI_ATTR(ANSI_INVERS);

    /* Cursor an Position zeile = 5 und spalte = 10 setzen */
    ANSI_CURPOS(5,10);

    /* Text ausgeben an zeile = 5 und spalte = 10 */
    TEXT(5,10);

    /* Cursor an Position zeile = 7 und spalte = 10 setzen */
    ANSI_CURPOS(7,10);

    /* Attribut normal anschalten */
    ANSI_ATTR(ANSI_NORMAL);
}
```

►► *Abb. 4d: ansi1.c nach Präprozessordurchlauf; Die Leerzeilen entstehen durch Entfernung der Kommentare.*

```
main ()
{

    printf ("\033[2J");


    printf ("\033[%um",7);


    printf ("\033[%u;%uH",5,10);


    printf ("Dieser Text beginnt an Zeile/Spalte %d/%d",5,10);


    printf ("\033[%u;%uH",7,10);


    printf ("\033[%um",0);
}
```

Die Präprozessordirektive #include

Die #include-Anweisung weist den Präprozessor an, die angegebene Datei in den Sourcecode einzulesen. So können Gruppen von #define-Anweisungen in Dateien ausgelagert werden, um das Programm übersichtlicher zu gestalten. Auch wir wollen dies tun. Erfassen Sie ein Headerfile – so werden in C die Dateien genannt, die mit Hilfe der #include-Anweisung eingelsen werden – mit dem Namen ckurs.h, löschen Sie die #define-Anweisungen im Programm ansi.c und fügen Sie folgende Zeile als erste Zeile des Programms ein:

```
#include  "ckurs.h"
```

Übersetzen Sie das Programm und führen Sie es aus. Das Programm aus *Abb. 4b* zeigt das Hea-

derfile ckurs.h und die #include-Anweisung. *Abb. 4d* zeigt das Sourcefile, nachdem der Präprozessor das Headerfile gelesen und die #define-Anweisungen ausgeführt hat.

1.3 printf und getchar

Da C über keine sprachimmanenten Möglichkeiten verfügt, Informationen auf dem Bildschirm auszugeben, gibt es die Funktion printf. Wir haben sie in den Beispielen bereits benutzt, ohne den genauen Aufbau der Funktion zu erklären. An dieser Stelle verweisen wir auf den Abschnitt Library Guide, in dem die Funktion printf im Detail erläutert wird.

Wenden wir uns jetzt getchar zu; getchar ist keine Funktion (s.a. Library Guide) sondern ein Makro, das in dem Headerfile stdio.h (wird zum Compiler mitgeliefert) definiert ist. Wenn Sie das Makro benutzen wollen, müssen Sie das Headerfile mit Hilfe einer #include-Anweisung einbinden. Wir wollen das Programm in Abb. 4 derart erweitern, daß nach der Textausgabe das Programm solange wartet bis der Benutzer die Eingabetaste betätigt. Nach dieser Bestätigung soll der Bildschirm wieder gelöscht werden. Versuchen Sie bitte, das Programm zu schreiben, ohne die *Abb. 5* anzusehen. Vergleichen Sie anschließend Ihre Version mit dem dort angegebenen Programm.

```
#include        <stdio.h>
#include        "ckurs.h"

main ()
{
    ANSI_CLR();
    ANSI_ATTR(ANSI_INVERS);
    ANSI_CURPOS(5,10);
    TEXT(5,10);

    getchar();

    ANSI_ATTR(ANSI_NORMAL);
    ANSI_CLR();
}
```

◂◂ *Abb. 5*

1.4 Operatoren

C verfügt über eine Vielzahl von Operatoren (über 30), die eine kompakte Programmierung ermöglichen. Operatoren in C verlangen einen, zwei oder drei Operanden. Operanden können auch Ausdrücke sein, so daß die Komplexität eines Ausdrucks beliebig komplex sein kann. Manche Operatoren haben je nach ihrem Kontext eine unterschiedliche Bedeutung, da der sichtbare ASCII-Zeichensatz nicht genügend eindeutige Operatoren zur Verfügung stellt. Wir werden an dieser Stelle nicht alle Operatoren behandeln. Im ersten Teil des C-Kurses sprechen wir die Arithmetik-, Vergleichs-, Zuweisungs- und Inkrementaloperatoren an.

1.4.1 Arithmetische Operatoren

Die arithmetischen Operatoren +, -, *, /, %, unterscheiden sich nicht von denen anderer Sprachen:

Addition	+	:	a= 5+3a= 8
Subtraktion	–	:	a= 5–3 a= 2
Division	/	:	a= 5/3 a= 1
Multiplikation	*	:	a= 5*3 a= 15
Modulo	%	:	a= 5%3a= 2

Interessant sind hier Division und Modulo. Der Modulooperator (%) ermittelt den ganzzahligen Rest einer Division. Die Division (/) ist auch für ganze Zahlen (Datentyp int) definiert. Das Ergebnis ist von dem gleichen Datentyp wie die Operatoren. Da 5 und 3 Integerkonstanten sind (siehe oben), ist auch das Ergebnis vom Datentyp int.

1.4.2 Zuweisungsoperatoren

Der Zuweisungsoperator kann mit den arithmetischen Operatoren verknüpft werden und erspart dann die Wiederholung der Variablen innerhalb der Zuweisung. Was in anderen Programmiersprachen als

```
zaehler= zaehler + 5
```

geschrieben werden muß, kann in C viel kürzer als

```
zaehler += 5;
```

codiert werden. Beachten Sie bitte, daß jede Anweisung mit einem Semikolon abgeschlossen werden muß.

1.4.3 Inkrement- und Dekrementoperatoren

Die Inkrement- und Dekrementoperatoren erhöhen (++) bzw. vermindern (--) ihren Operanden um eins. Statt

```
a= a+1
```

schreibt man

```
a++
```

Hier kann der Compiler effizienten Code generieren, indem er statt einer Addition eine Inkrementoperation ausführt. Der Inkrementoperator kann auch innerhalb beliebiger Ausdrücke, zum Beispiel bei der Indizierung eines Arrays, verwendet werden. Die Inkrement- und Dekrementoperatoren können als Post- und als Pränotation verwendet werden. Die Postnotation (a++) bewirkt, daß erst der aktuelle Wert von a im Gesamtausdruck verarbeitet und anschließend a inkrementiert wird. Bei der Pränotation (++a) wird a erst inkrementiert und dann im Ausdruck verwendet:

```
a= 2;
b= a++; /* b= 2, a= 3 */
c= ++a; /* c= 4, a= 4 */
```

1.4.4 Vergleichs- und logische Operatoren

Die Vergleichsoperatoren (>, >=, <, <=, ==, !=) werden für die Programmablaufsteuerung benutzt (s.u.). Die Bewertung ergibt 0, wenn die Bedingung nicht erfüllt ist, anderenfalls 1. Beispiel:

5<4 :	0
6>=-3 :	1
3==7 :	0
-10!=3 :	1

Beachten Sie bitte, daß der Gleichheitsoperator mit zwei Gleichheitszeichen geschrieben wird!

Die logischen Operatoren (&& = AND, || = OR, ! = NOT) werden dazu benutzt, Vergleiche in Bedingungen zu verbinden:

```
a==b && a==c
```

liefert den Wert 1, wenn a, b und c gleich sind. ! (NOT) negiert den Wert einer Anweisung, so daß

```
a==0 und !a
```

identisch sind.

1.5 Programmsteuerung (if, for, switch)

Die Anweisungen für die Programmsteuerung dienen dazu, die festgelegte Reihenfolge der sequentiellen Abarbeitung der Anweisungen zu ändern.

1.5.1 Die if-Anweisung

Als erstes ändern wir unser Programm, indem wir abfragen, in welche Klasse das erste eingelesene Zeichen in unserem Programm gehört. Dazu definieren wir folgende Klassen:

- Großbuchstaben
- Kleinbuchstaben
- Ziffern
- Sonderzeichen

Das if-Statement führt die auf die Bedingung folgende Anweisung aus, wenn das Ergebnis der Bewertung 1 ist, ansonsten wird die Anweisung nach dem else ausgeführt. Folgen nach der Bedingung mehrere Anweisungen, müssen diese innerhalb eines Blocks stehen. Sehen Sie sich als Beispiel das Programm der *Abb. 6* an.

►► *Abb. 6: Programm mit if-Anweisung*

```
#include        <stdio.h>
#include        "ckurs.h"


main ()
{
    int c;

    ANSI_CLR();
    ANSI_ATTR(ANSI_INVERS);
    ANSI_CURPOS(5,10);
    TEXT(5,10);

    ANSI_CURPOS(7,10);
    ANSI_ATTR(ANSI_NORMAL);

    c= getchar();

    ANSI_CLR();

    if (c>='A' && c<='Z')  /* Großbuchstabe */
            printf ("Großbuchstabe\n");
    else
            if (c>='a' && c<='z')  /* Kleinbuchstabe */
                printf ("Kleinbuchstabe\n");
            else
                if (c>='0' && c<='9')  /* Ziffer */
                        printf ("Ziffer\n");
                else
                        printf ("Sonderzeichen\n");
}
```

Die Bedingung

```
if (c>='A' && c<='Z')
```

liefert den Wert 1, wenn c (Funktionswert von getchar()) zwischen 'A' und 'Z' liegt. Der Ausdruck enthält Zeichenkonstanten, Vergleichsoperatoren und logische Operatoren. Ist der Wert der Bedingung 0, wird die else-Anweisung ausgeführt.

1.5.2 Die for-Schleife

Die for-Schleife in C besteht aus einem Schleifenkopf und einem Schleifenrumpf. Der Schleifenkopf enthält das Schlüsselwort for, eine Initialisierung, einen Kontrollteil und Modifizierung, alles durch ; getrennt:

```
for (Initialisierung; Bedingung; Modifizierung)
{
    /* Dies ist der Schleifenrumpf */
}
```

Der Schleifenrumpf muß nur mit {} umschlossen werden, wenn mehr als eine Anweisung folgt. Die Anweisung(en) werden ausgeführt, solange die Bedingung einen Wert ungleich 0 liefert. Vor dem ersten Durchlauf wird die Initialiserung ausgeführt (mehrere Anweisungen werden durch "," getrennt) und die Bedingung getestet. Beispiel:

```
    for (i= 0; i<10; i++)
        printf ("Dies ist Schleifendurchlauf %d \n", i);
```

Die Schleife wird zehnmal durchlaufen und der aktuelle Schleifendurchlauf wird angezeigt. Die Bedingung (i<10) gibt an, wie lange die Schleife durchlaufen werden soll (solange i kleiner als 10 ist). Der Initialisierungsteil setzt i vor dem ersten Durchlauf auf den Wert 1 (i= 1). Der Modifikationsteil wird nach jedem Schleifendurchlauf ausgeführt. Er dient in unserem Beispiel dazu, die Variable i um 1 zu inkrementieren.

Ändern Sie bitte unser Programm aus Abb. 6 so ab, daß der Text fünfmal untereinander ausgegeben wird.

1.5.3 Die switch-Anweisung

Die switch-Anweisung erlaubt in C die Programmierung einer Fallabfrage. Für eine Bedingung können für verschiedene Fälle unterschiedliche Anweisungen ausgeführt werden. In *Abb. 7* wurde das Programm der Abb. 6 dahingehend geändert, daß abgefragt wird, ob eine Ziffer zwischen 1 und 3 eingegeben wurde. Wenn ja, wird der um die Anzahl der Zeilen nach unten verschoben, erneut ausgegeben. Beachten Sie bitte die Schlüsselwörter case und break. Ein Fall wird mit case, einer Konstanten und dem Doppelpunkt : definiert. Er endet mit break. Wird dieses break vergessen, wird die nächste Anweisung des folgenden Falls ausgeführt!

Die default-Anweisung wird ausgeführt, wenn keiner der angegeben Fälle zutrifft.

Sie haben in diesem Abschnitt die wichtigsten Operatoren und Kontrollstrukturen kennen-

gelernt. Sie wissen, wie man in C Konstanten definiert und diese in #define-Anweisungen und Headerfiles (#include-Anweisung) auslagert. Sie haben erfahren, daß C keine Schlüsselwörter für die Ein- und Ausgabe bereithält, sondern daß dafür die Funktionen printf und getchar verwendet werden können. Im folgenden Abschnitt werden printf und getchar im Detail behandelt.

```
#include        <stdio.h>
#include        "ckurs.h"

main ()
{
    int c, i;

    ANSI_CLR();
    ANSI_ATTR(ANSI_INVERS);
    ANSI_CURPOS(5,10);
    TEXT(5,10);

    ANSI_CURPOS(7,10);
    ANSI_ATTR(ANSI_NORMAL);

    c= getchar();

    i= 6;

    switch (c)
    {
        case '1':   ANSI_CURPOS(6,10);
                    break;

        case '2':   ANSI_CURPOS(7,10);
                    i++;
                    break;

        case '3':   ANSI_CURPOS(8,10);
                    i+= 2;
                    break;

        default:    printf ("Keine Ziffer zwischen 1 und 3"
                            "eingegeben\n");
                    i= 0;
                    break;
    }

    if (i)
        TEXT(i,10);
}
```

◂◂ *Abb. 7: Programm mit switch-Anweisung*

2. Library Guide

Der Library Guide des C-Compilers ist die Beschreibung aller vom Compiler angebotenen Funktionen. Wie oben bereits näher ausgeführt, bestimmt die mitgelieferte C-Bibliothek die Güte des Compilers. Für den MS-C-Compiler werden ca. 600 und für den Quick-C Compiler 400 Funktionen mitgeliefert. Da die Funktionen eines C-Compilers nicht genormt sind (ein ANSI-Entwurf liegt zwar vor, eine Norm wurde bis dato jedoch nicht verabschiedet), kann es bei Portierungen zu Problemen kommen, wenn benutzte Funktionen in der C-Bibliothek des anderen Compilers nicht vorhanden sind. Dies wird innerhalb dieses Kapitels bei Benutzung von nicht portablen Funktionen immer gekennzeichnet. So sind beispielsweise sämtliche Grafikfunktionen nur unter MS-DOS und OS/2 verfügbar.

Innerhalb des Library Guides des MS-C-Compilers 5.1 werden die Funktionen alphabetisch aufgeführt. Zu jeder Funktion werden die Schnittstelle (inklusive der verwendeten Headerfiles), eine genaue Beschreibung der Funktion, die Funktionswerte, Hinweise auf ähnliche Funktionen und ein Beispiel zur Anwendung der Funktion in Form eines C-Programms beschrieben.

Der Library Guide des Quick-C Comilers ist etwas anders aufgebaut. Dort werden die Funktionen nicht alphabetisch aufgeführt, sondern in Funktionsgruppen unterteilt. Es werden die benutzten Headerfiles, die Schnittstelle und die Funktionswerte beschrieben.

In den Beispielen des ersten Seminarteils wurden die Funktionen printf() und getchar() angesprochen.

2.1 Die Funktion printf

Schnittstelle

```
#include <stdio.h>

int printf (format [, argument] ...)
const char  *format;
```

Die eigentliche Aufgabe der Funktion printf() ist es, einen String auf dem Bildschirm auszugeben. In Abb. 1 wird printf() dazu verwendet, Informationen auf dem Bildschirm darzustellen.

Um auch Werte von anderen Datentypen (int, long, float, double) anzeigen zu können, kann der String mit Formatangaben belegt werden. Formatangaben fangen immer mit dem %-Zeichen an, gefolgt von dem Format selbst. In Abb. 2 wurden die Bytelängen (int) der einzelnen Datentypen ausgegeben. Für int-Werte wird die Formatangabe %d benutzt. Für jede Formatangabe innerhalb des String muß der Funktion ein Parameter übergeben werden. Die Funktion hat folglich eine variable Anzahl von Parametern und zwar immer »Anzahl der Formate des Formatstrings plus 1 (der String selber)«.

Soweit zum einfachen Verständnis der Funktion. Im folgenden wird der Formatstring systematisch in seine kombinierbaren Bestandteile zerlegt, damit Sie sich einen Überblick über die Mächtigkeit der Funktion printf() verschaffen können. In der Praxis wird man dann mit einigen wenigen Formatangaben auskommen.

Der Aufbau des Formatstrings

printf() erhält als ersten Parameter einen Formatstring. Der Formatstring besteht, wie oben bereits erwähnt, aus normalen Zeichen und den Formatangaben. Für jede Formatangabe muß eine Variable (durch Komma getrennt) nach dem Formatstring angegeben werden.

Ist die Anzahl der dem Formatstring folgenden Parameter ungleich der Anzahl der Formatangaben in dem Formatstring, so werden entweder weniger Variablen formatiert (Anzahl der Parameter größer) oder es kommt zu undefinierten Ausgaben (Anzahl der Parameter kleiner).

Der Formatstring wird wie folgt zerlegt:

```
% [Ausrichtung] [Breite] [.Genauigkeit] [F|N|h|l|L] Typ
```

Den einzelnen Teilen des Formatstrings kann jeweils ein Wert aus den folgenden Tabellen zugeordnet werden. Die Teile in den eckigen Klammern sind optional anzugeben.

Typ
Als erstes werden die verschiedenen Typen definiert, da in der weiteren Erläuterung immer wieder Bezug darauf genommen wird.

Typ	Datentyp	Ausgabe
d,i	int	numerisch mit Vorzeichen (-32.767 .. +32.767)
u	int	numerisch ohne Vorzeichen (0 .. 65535)
o	int	oktal (0 .. 177777)
x	int	hexadezimal (benutzt »abcdef«, 0 .. ffff)
X	int	hexadezimal (benutzt »"ABCDEF«, 0 .. FFFF)
f	double	gebrochener Wert der Form [-]dddd.dddd
e	double	Exponentialdarstellung der Form [-]d.dddd e [-/+]ddd
E	double	wie e, jedoch wird e als E ausgegeben
g	double	wird ausgegeben wie f oder e, je nachdem welche Ausgabe kürzer ist
G	double	wie g, jedoch bei E-Formatierung Ausgabe mit E
c	char	ein einzelnes Zeichen
s	char *	Ausgabe eines Strings vom übergebenen Pointer bis zur Stringendekennung ('\0')
p	Zeiger	gibt die Adresse des Zeigers in der Form xxxx.yyyy aus, wobei xxxx das Segment und yyyy der Offset innerhalb des Segmentes ist
%	-	Ausgabe des Prozentzeichens

Ausrichtung

Zchn	Bedeutung	Vorbelegung, wenn nicht gesetzt
-	linksbündig innerhalb der gesetzten Feldbreite	rechtsbündig
+	Ausgabe eines Pluszeichens vor der Zahl	kein Pluszeichen
' '	Ausgabe eines Leerzeichens vor der Zahl	kein Leerzeichen
#	bei Verwendung mit Typ o, x oder X: keine Ausgabe	jedem Wert ungleich 0 wird ein 0, 0x oder 0X vorangestellt
	bei Verwendung mit Typ e, E, f: Ausgabe des Dezimalpunktes nur wenn Nachkommastellen vorhanden sind	die Ausgabe enthält immer einen Dezimalpunkt
	bei Verwendung mit Typ g oder G: wie oben; wie bei Typ e, E oder f, jedoch werden hier auch die Nullen nach dem Dezimalpunkt nicht abgschnitten	Nullen nach dem Dezimalpunkt werden abgeschnitten
	Bei den Typen c, d, i, u oder s wird dieses Formatzeichen ignoriert	

Breite und Genauigkeit
Die Breite gibt die minimale Anzahl von Zeichen an, die ausgegeben werden sollen. Wenn die Ausgabe kleiner ist als die Breite, so werden – in Abhängigkeit von der Ausrichtung – links oder rechts Leerzeichen eingefügt. Beginnt die Breitenangabe mit einer 0, so werden anstatt der Blanks Nullen eingefügt.

Die Breite ist also nicht die absolute Breite der Ausgabe, sondern die Anzahl der Zeichen, die mindestens ausgegeben werden sollen. Die Ausgabe wird nie abgeschnitten.

Daneben existiert eine Ausnahme. Die Breite kann auch ein Stern * sein. In diesem Fall wird der korrespondierende int-Wert der Parameterliste als Feldbreite herangezogen.

Die Genauigkeit ist abhängig von dem verwendeten Typ.

▸▸ *Abb. 8: printf-Beispiele*

```
main ()
{
    int i;
    int breite;
    int genau;

    i= 12345;
    breite= 10;
    genau= 7;

    printf ("<%d>\n", i);
    printf ("<%-d>\n", i);
    printf ("<%10d>\n", i);
    printf ("<%-10d>\n", i);
    printf ("<%010d>\n", i);
    printf ("<%-010d>\n", i);
    printf ("<%10.6d>\n", i);
    printf ("<%-10.6d>\n", i);
    printf ("<%*.6d>\n", breite, i);
    printf ("<%-*.*d>\n", breite, genau, i);
}
```

Geben Sie das Programm der *Abb. 8* ein und sehen Sie sich die Ergebnisse an. Bei diesem Programm wurde bereits mit der Genauigkeit gearbeitet. Diese gibt bei den Typen d, i, u, o, x, und X an, wieviel Ziffern mit führenden Nullen innerhalb der Breite ausgegeben werden sollen.

Die Genauigkeitsangabe in Verbindung mit den Typen e, E und f bestimmt die Anzahl der Nachkommastellen, wobei die letzte dargestellte

Ziffer gerundet wird. Hier wird als Vorbelegung eine 0 gesetzt. Die Typen g und G interpretieren die Genauigkeit als maximale Anzahl signifikanter Ziffern. Wird nichts angegeben, werden alle Ziffern ausgegeben.

Für Strings (Typ s) bestimmt die Genauigkeit die maximale Anzahl der Zeichen, die ausgegeben werden sollen. Ansonsten wird bis zum nächsten Stringendekennzeichen ('\0') ausgegeben. Ändern Sie das Programm der Abb. 8 dahingehend ab, daß Sie sich einen String

```
#define  TEXT  "Text für die Formatierung mit printf"
```

definieren und die Variable i durch TEXT sowie %d durch %s ersetzen.

F, N, h, l, L

Diese Angaben innerhalb des Formatstrings ändern den erwarteten Datentyp der Parameterleiste.

F und N ändern die Pointerart. F bestimmt einen far Pointer und N einen near Pointer. Diese Formatierungsart ist nicht kompatibel zum ANSI-Standard.

Der Prefix h wird dazu benutzt, der Formatierungsroutine mitzuteilen, daß für die Typen d, i, o, x und X der Parameter nicht vom Datentyp int, sondern vom Datentyp short int ist. Gleiches gilt für u (unsigned short int).

l ändert die Parameter der Typen d, i, o, x und X von int auf long int. Für den Typ u wird mit lu ein Parameter vom Datentyp unsigned long int erwartet.

L kann nur im Zusammenhang mit den Typen e, E, f, g und G benutzt werden. Es wird dann anstatt eines double- ein long double-Parameter erwartet.

Die Funktion printf() wird auf den Seiten 456ff. der Run-Time Library Reference beim Microsoft 5.1 Compiler und auf der Seite 328 beim Quick-C Compiler 2.0 beschrieben.

Funktionswert

Die Funktion liefert die Anzahl der ausgegebenen Zeichen.

Verwandte Funktionen

sprintf: sprintf stellt das Ergebnis nicht auf dem Bildschirm dar, sondern in einem Buffer, der der Funktion übergeben wird:

```
int sprintf (buffer, format [, argument] ...)
char        *buffer;
const char  *format;
```

Dabei ist zu beachten, daß der Datenbereich, auf den buffer zeigt, groß genug sein muß, um alle Zeichen, die sprintf erzeugt, aufnehmen zu können.

fprintf: Bei fprintf wird das Ergebnis in eine Datei geschrieben. Die Dateioperationen werden in einem weiteren Teil des C-Kurses behandelt.

vprintf, vsprintf, vfprintf: Der Unterschied zu den Funktionen printf, sprintf und fprintf besteht darin, daß die Funktionen anstatt der Parameterliste einen Pointer auf eine Liste von Argumenten übergeben bekommen. Diese Funktionen sind hier nur der Vollständigkeit halber aufgeführt; mit Ihrem bisherigen Wissen über C kann noch nicht mit diesen Funktionen gearbeitet werden. Wir verweisen daher auf eine spätere Folge dieses Seminars.

scanf: scanf() ist das Analogon zu printf(). Während printf() auf dem Bildschirm formatiert, liest scanf() von der Tastatur Zeichen ein. Diese Zeichen werden auf die Parameterliste übertragen. Da Sie hierzu wissen müssen, wie Sie mit Zeigern auf Variablen arbeiten, wird nicht an dieser Stelle, sondern erst im Kapitel über Zeiger ein Beispiel folgen.

2.2 Die Funktion getchar

Schnittstelle

```
int  getchar()
```

Die Funktion wird dazu benutzt, ein Zeichen von der Tastatur einzulesen und dies als Funktionwert an die aufrufende Funktion zu übergeben, wie dies in Abb. 6 und Abb. 7 gemacht wird.

getchar() ist eigentlich keine Funktion. getchar() wird in dem Headerfile stdio.h mit Hilfe folgender Präprozessoranweisung umgesetzt:

```
#define  getchar()  getc(stdin)
```

Getc() selbst ist auch ein Makro (so nennt man auch Umsetzungen des Präprozessors, die quasi als Funktionen benutzt werden). Die Umsetzung hier anzuführen, würde im Moment nur verwirren; deshalb wird hier nur die Funktionsweise beschrieben.

Stdin ist ein Filepointer (Dateizeiger), in den Dateiinformationen geschrieben werden. Ein Filepointer wird beim Öffnen der Datei erzeugt. Er zeigt auf eine Datenstruktur, die Informationen über die Datei enthält (Zeiger auf nächstes Zeichen der Datei, Open-Flags: readonly, writeonly, read/write, Dateinummer des Betriebssystems, etc.). Auch der Bildschirm wird in C wie eine Datei behandelt. Beim Starten des Programms werden – noch bevor die main()-Funktion aufgerufen wird – folgende Dateien geöffnet:

stdin	Standard Input
stdout	Standard Output
stderr	Standard Error
stdaux	Standard Auxilary
stdprn	Standard Printer

Als C-Programmierer können Sie, ohne vorher eine Datei geöffnet zu haben, von der Tastatur Zeichen einlesen (stdin) und auf dem Bildschirm Informationen ausgeben (stdout). Die Buffer stdaux und stdprn werden nur vom Microsoft C-Compiler geöffnet. Andere C-Compiler-Implementierungen verwenden nur die ersten drei Buffer.

Wie der Buffer aufgebaut ist, erfahren Sie in einem der nächsten Seminarteile.

Die Funktion überträgt die Zeichen erst, wenn ein Zeilenendekennzeichen eingegeben wurde. Unter MS-DOS ist dies die Returntaste ('\r').

Verwandte Funktionen

fgetc, fgetchar: Diese Funktionen arbeiten auf Dateien. Es wird das nächste Zeichen von einer Datei eingelesen. Sie haben die gleiche Funktionanlität wie getc() und getchar().

getch, getche: Diese Funktionen übergeben das Zeichen sofort nach dem es eingegeben wurde. Die Funktion getch() übergibt das Zeichen nur an das Programm, getche() (e=echo) läßt es zusätzlich auf dem Bildschirm erscheinen.

putc, putchar: putc() und putchar() schreiben ein Zeichen auf den spezifizierten Buffer (s.o.). Damit können Zeichen an der aktuellen Position auf dem Bildschirm ausgegeben werden. *Abb. 9* enthält ein Programmbeispiel, das alle Zeichen von der Tastatur einliest und anschließend nur die Buchstaben zwischen A und M auf dem Bildschirm ausgibt. Die Schleife wird beendet, wenn ein X eingegeben wird oder ein Fehler beim Einlesen der Zeichen aufgetreten ist.

▸ Abb. 9: Beispiel mit getch

```
#include  <stdio.h>
#include  <conio.h>

main ()
{
     char  ch;

     while ( ( ch= getch() ) != EOF)
     {
         if (ch>='A' && ch<='M')
             putchar (ch);

         if (ch=='X')
             break;
     }
}
```

Beachten Sie bitte dabei, daß die Funktion getch() verwendet wird, die das eingegebene Zeichen nicht auf dem Bildschirm darstellt.

▸ Abb. 10: Umwandlung von eingegebenen Zeichen; Bitte beachten Sie die Zeile ch-= 'A' - 'a'. In C können Sie den Abstand zwischen den Groß- und Kleinbuchstaben durch die Differenz 'A' - 'a' bestimmen.

```
#include  <stdio.h>

main ()
{
    char  ch;

    while ( ( ch= getchar() ) != EOF)
    {
        if (ch>='A' && ch<='M')
            putchar (ch);

        if (ch>='a' && ch<='m')
        {
            ch-= 'a' - 'A';
            ungetc(ch, stdin);
        }

        if (ch=='X')
            break;
    }
}
```

ungetc: Die Funktion ungetc() wird dazu benutzt, um bereits eingelesene Zeichen wieder in den Dateibuffer zu stellen. Sie wird in *Abb. 10* in Ergänzung zum Beispiel in Abb. 9 dazu verwendet, alle Kleinbuchstaben in Großbuchstaben umzuwandeln (beachten Sie bitte, daß in dieser Funktion mit getchar() die Zeichen eingelesen werden!).

Sie haben nun die Funktionen printf() und getchar() kennengelernt. Das nächste Kapitel beschäftigt sich dem MS C-Compiler Version 5.1. Dort werden wichtige Optionen besprochen und die Arbeitsweise des Compilers erklärt.

3. Quick-C und MS C-Compiler 5.1

Nachdem in den Beispielen die Entwicklungsumgebung Quick-C zur Eingabe und Übersetzung verwendet wurde, wird in diesem Kapitel erläutert, wie mit dem MS C-Compiler Version 5.1 gearbeitet wird.

3.1 Arbeitsweise

Der MS C-Compiler übersetzt ein im Editor erstelltes C-Programm in ein Objektmodul bzw. erstellt ein ausführbares Programm (s. Optionen innerhalb dieses Kapitels). Die Übersetzung läuft in drei verschiedenen Phasen ab:

Präprozessor: Der Präprozessor liest alle angegebenen Headerfiles ein und ersetzt die #defines und Makros durch die definierten Werte. Sie können sich eine solche Datei erzeugen lassen, wenn Sie die Option /P verwenden. Die vom Präprozessor erzeugte Ausgabe wird in die Datei fn.i gestellt, wenn fn.c der Name des C-Programms ist.

Syntaxchecker: Der Syntaxchecker bekommt als Eingabe die vom Präprozessor erstellte Datei. Die Art der Syntaxüberprüfung kann mit der /W-Option angegeben werden (s.u.).

Codegenerierung: Die syntaktisch richtige Datei wird dann dem Codegenerierungsmodul übergeben. Auch hier können Sie die Art der Codegenerierung beeinflussen. Alle Optionen, die mit /G beginnen, werden dazu verwendet bestimmten Maschinencode zu erzeugen (8086, 80286, etc.). Optionen, die mit /O anfangen, verändern den Code in Hinsicht auf Optimierungen (/Os für size = Größe, /Ot für time = zeit). Alle mit /A anfangenden Optionen bestimmen das verwendete Speichermodell und damit ebenfalls die Codegenerierung.

Die in diesem Seminarteil angegebenen Beispiele können auch mit dem MS C-Compiler Version 5.1 übersetzt werden. Der Aufruf des Compilers erfolgt mit:

```
cl beispiel.c
```

Dieser Befehl übersetzt das Programm beispiel.c und erstellt das ausführbare Programm beispiel.exe. Wenn Sie lediglich ein Objektmodul erzeugen wollen, müssen Sie die Option /c hinzufügen:

```
cl /c beispiel.c
```

Die beiden anderen Optionen /AM sowie /W3

werden in den folgenden beiden Abschnitten »Speichermodell« und »Warnungsstufe« erläutert.

Mit dem Aufruf

```
cl /HELP
```

werden auf dem Bildschirm alle möglichen Optionen und eine Kurzhilfe zu jeder Option ausgegeben.

3.2 Speichermodelle

Aufgrund der Adressierungsart des 80x86-Prozessors (Segment und Offset) gibt es mehrere Möglichkeiten, Objektcode zu generieren. Je nachdem, ob das Segment konstant oder variabel ist, können maximal 64 Kbyte oder mehr angesprochen werden. Diese Besonderheit des Prozessors nennt man auch Speichermodell. Weil für die Codegenerierung jeweils zwei Segmente verwendet werden (Daten- und Codesegment), kann der MS C-Compiler für insgesamt vier (2 mal 2) verschiedene Speichermodelle Objektcode erzeugen. Die Speichermodelle werden mit der /A-Option – gefolgt von dem Speichermodell – angegeben:

/AS (Small Memory Model, Default-Einstellung)
/AM (Medium Memory Model)
/AC (Compact Memory Model)
/AL (Large Memory Model)

Die Speichermodelle unterscheiden sich hinsichtlich der Größe des Datenbereichs und der Größe des Programmcodes. In der folgenden Übersicht werden für jedes Speichermodell die Segmentdefinitionen angegeben:

/AS: Datenbereich und Programmcode können je 64 Kbyte nicht überschreiten (Daten- und Codesegment konstant).

/AM: Der Datenbereich ist maximal 64 Kbyte groß und der Programmcode kann theoretisch beliebig (MS-DOS Grenze 640 Kbyte) groß sein (Datensegment konstant und Codesegment variabel).

/AC: Datenbereich ist beliebig (MS-DOS Grenze 640 Kbyte) groß und der Programmcode kann 64 Kbyte nicht überschreiten.

/AL: Der Datenbereich und der Programmcode können beliebig (MS-DOS Grenze 640 Kbyte) groß werden, jedoch sind einzelne Varibleń auf 64 Kbyte begrenzt.

Neben diesen vier Speichermodellen existiert noch das Huge-Speichermodell (Option /AH). Es entspricht dem des Large-Modells, nur daß auch einzelne Datenelemente (beispielsweise Felder) größer als 64 Kbyte werden können.

Die Art des Speichermodells hat auch Auswirkungen auf die Geschwindigkeit des Programms. Bei konstanten Segmenten muß das Segmentregister, das zur Adressierung von Programmcode und Datenbereich herangezogen wird, nur bei Programmstart geladen werden (Small-Modell). Sind die Segmente variabel (Large-Modell), so müssen für jede Funktion und für jede Variable die entsprechenden Segmentregister geladen werden.

3.3 Warnungen

Der MS C-Compiler bietet insgesamt vier Warnungsstufen an. Diese können mit der Option /W und Angabe der Warnungsstufe eingeschaltet werden. Warnungen dienen dazu, die Syntax zu überprüfen und bei Funktionsaufrufen die aktuellen mit den formalen Parametern zu prüfen. Die formalen Parameter können in sogenannten Funktionsprototypen auch für Funktionen definiert werden, die sich nicht im aktuell übersetzten Sourcecode befinden. Die aktuellen Parameter werden beim Aufruf der Funktion mit den formalen verglichen. Abweichungen werden in Form von Fehlermeldungen dokumentiert. Im Anhang E (Error Messages) des User's Guide werden alle Fehler, die der Compiler beim Übersetzen meldet, aufgeführt. Warnungen beginnen mit C4. Sie enthalten neben dem Text auch einen Hinweis, ab welcher Warnungsstufe dieser Fehlertext ausgegeben wird. In dieser Folge wird mit der Warnungsstufe 1 gearbeitet, da für die Arbeitsweise der Warnungsstufen 2 und 3 ein detailliertes Verständnis von Funktionsprototypen erforderlich ist, die erst in der nächsten Folge besprochen werden.

/W0 oder /w: mit diesen beiden Optionen werden alle Warnungen unterdrückt.

/W1 ist die Voreinstellung, wenn keine /W-Option angegeben wird. Innerhalb dieser Warnungsstufe werden fast alle Warnungen ausgegeben.

/W2 überprüft zusätzlich, ob

- für aufgerufene Funktionen deren Funktionswerte angegegeben worden sind,
- innerhalb einer Funktion, die nicht den Datentyp void hat, eine return-Anweisung steht,
- bei Zuweisungen zwischen verschiedenen Datentypen Informationen verlorengehen können. Das in *Abb. 11* angegebene Programmbeispiel erzeugt beim Übersetzen eine eine Warnung, daß bei der Zuweisung Informationen verlorengehen.

```
main ()
{
    char  ch;
    int   i;

    i= 10;

    ch= i;
}
```

◂◂ *Abb. 11: Warnungsstufe 2; Compilerausgabe: test.c(8): C4051 data conversion*

/W3 überprüft bei Funktionsaufrufen die aktuellen mit den formalen Parametern (Funktionsprototypen). Diese Überprüfung gehört nicht zum C-Compiler nach /1/. Sie wurde von Microsoft implementiert, damit ein häufiger Fehler bei C-Programmen (nicht korrekte Versorgung der

Funktionen mit Parameter) vom Compiler überprüft werden kann. Funktionsprototypen müssen vor dem ersten Aufruf der Funktion definiert werden. Die Formalparameterdefinition wird bei jedem Aufruf der Funktion innerhalb des Programms mit den aktuell übergebenen Parametern verglichen. Wird eine Abweichung festgestellt, gibt der Compiler eine entsprechende Meldung aus. Funktionsprototypen werden in der nächsten Folge detailliert besprochen.

4. Aufgaben

Im folgenden werden Übungsaufgaben gestellt, die anhand des behandelten Sprachumfangs implementiert werden können. Die Musterlösungen werden in der nächsten Folge veröffentlicht.

4.1 Rahmenprogramm

Erstellen Sie ein Programm, das einen einfachen oder doppelten Rahmen zeichnet. Sie können mit Hilfe der Zeichenkonstanten ('\nnn') die Blockgrafik des IBM-Zeichensatzes benutzen.

4.2 Mehrfachselektion (check box)

Schreiben Sie ein Programm, daß folgende Texte auf dem Bildschirm ausgibt:

```
Beantworten Sie bitte folgende Fragen:


In der Programmiersprache C

[ ] endet jede Anweisung mit einem Semikolon
[ ] gibt es Schlüsselwörter für Ein- und Ausgabe
[ ] beginnen Felder bei 0
[ ] ist Gleichheitsoperator "="
[ ] sind die Anweisungen a=a+3 und a+=3 identisch

Leertaste = Auswahl
TAB       = nächstes Feld
Eingabe   = alle Fragen beantwortet
```

Der Benutzer soll die gestellten Fragen beantworten. Dazu wird der Cursor beim Programmstart auf die Position zwischen die ersten beiden eckigen Klammern gestellt. Der Benutzer hat dann die Möglichkeit, durch Eingabe der Leertaste die Frage anzukreuzen. Es erscheint ein 'X'. Um zu der nächsten Frage zu gelangen, muß er die Tab-Taste drücken. Wird in einem angekreuzten Feld die Leertaste gedrückt, soll das 'X' durch ein Leerzeichen ersetzt werden. Wenn der Benutzer mit der Benatwortung der Fragen fertig ist, gibt er die Eingabetaste ein. Danach soll die Meldung erscheinen, ob der Benutzer die Fragen richtig beantwortet hat.

Hinweis: Um die Aufgabe übersichtlich zu lösen, sollte ein Feld aus int-Werten definiert werden, in dem vermerkt wird, ob eine Möglichkeit an- oder ausgeschaltet ist. Verwenden Sie die im Abschnitt Library Guide angesprochene Funktion getch zum Einlesen von Zeichen. Benutzen Sie zur Positionierung die im Abschnitt Sprachelemente definierten ANSI-Sequenzen.

4.3 Einfachselektion (radio buttons)

Ändern Sie das Programm der Aufgabe 4.2 so, daß nur noch eine Antwort wählbar ist. Ersetzen Sie die eckigen durch runde Klammern und benutzen Sie als Markierungszeichen den ASCII-Wert '\371' (oktal).

Wenn der Benutzer eine Antwort ankreuzt, soll automatisch eine zuvor gewählte Alternative gelöscht werden.

4.4 Dialogbox

Kombieren Sie die drei zuvor gelösten Aufgaben zu einer Dialogbox. Eine Dialogbox hat eine doppelte Umrandung. Die Überschriften werden durch eine einfache Linie von den Selektionsfeldern getrennt.

5. Zusammenfassung und Ausblick

In dieser Folge haben wir wesentliche Grundelemente der Sprache C kennengelernt und an Beispielen zur Ein-/Ausgabe demonstriert.

Ein C-Programm besteht aus einer beliebigen Anzahl von Funktionen, von denen mindestens eine Funktion main heißen muß, die den Programmstart markiert. C kennt die elementaren Datentypen char, int, short, long, float und double. Konstanten haben in C immer eine Typbindung. Wir haben String-, Character-, Aufzählungs- und Integerkonstanten besprochen. Ein wesentlicher Bestandteil der Sprache C ist der Präprozessor. Er führt Textersetzungen vor der eigentlichen Übersetzung durch. Wir haben die Präprozessoranweisungen #define und #include kennengelernt.

In der nächsten Folge werden wir uns mit den restlichen Operatoren (Bit- und Boolschen Operatoren) auseinandersetzen. Es wird ein Taschenrechner entwickelt, der neben den gängigen mathematischen Operationen auch Umrechnungen zwischen verschiedenen Zahlensystemen beherrscht. Um die Daten einzugeben, entwickeln wir eine Funktion zur kontrollierten und typgenauen Dateneingabe. Wir besprechen die Definition von Funktionen und die Handhabung von Prototypen und erläutern den Aufbau eines C-Programms aus mehreren Sourcefiles sowie die Handhabung des Linkers.

Innerhalb des Kapitels Library Guide haben Sie die Funktionen bzw. Makros printf() und getchar() sowie deren verwandte Funktionen kennengelernt. Anhand einiger Beispiele wurde die Anwendung verdeutlicht. Im nächten Teil erfahren Sie mehr über String- und Datentypumwandlungsfunktionen.

Im Kapitel über den MS C-Compiler wurde die Arbeitsweise (Präprozessor, Syntaxüberprüfung, Codegenerierung) erläutert und wichtige Optionen erklärt. Der nächste Teil beschäftigt sich mit dem Environment des Compilers und beschreibt die getrennte Compilierung von C-Programmen.

6. Literaturverzeichnis

/1/ Kerninghan, Brian W., Ritchie, Dennis M.: The C Programming Language, Prentice-Hall, New Jersey.

/2/ Kerninghan, Brian W., Ritchie, Dennis M.: The C Programming Language, Second Edition, Prentice-Hall, New Jersey.

/3/ Kelley, Al, Pohl, Ira:
An Introduction to Programming in C, The Benjamin/Cummings Publishing Company, California.

/4/ Feuer, Alan R.:
The C Puzzle Book, Prentice-Hall, New Jersey.

/5/ Schildt, Herbert: C: The Complete Reference, McGraw-Hill, California.

/6/ Microsoft Corporation: C For Yourself, USA/Canada.

/7/ Microsoft Corporation: Run-Time Library Reference.

Die aufgeführten Bücher sind zum Teil auch ins Deutsche übersetzt worden.

7. Die Autoren

Dipl.-Kfm. Jochen Witte, 1960 in Dortmund geboren, studiert an der TU Berlin Betriebswirtschaft und im Laufe des Studiums sowie im Rahmen seiner Diplomarbeit zur Informatik gefunden, um sich dort mit der Anwendungsprogrammierung in C zu beschäftigen. Er arbeitet seit Anfang 1988 bei der BKS Software Entwicklungs GmbH im Vertrieb und ist verantwortlich für das Schulungsprogramm (C, XENIX, BKS-STOP).

Dipl.-Inf. Rainer Kreutzer, 1961 in Berlin geboren, ist Mitgründer der BKS GmbH und als geschäftsführender Gesellschafter verantwortlich für Produktentwicklung sowie Projektleitung bei der Erstellung von Individualsoftware.

Das 1984 gegründete Softwarehaus BKS entwickelt professionelle C-Tools für alle PC-Betriebssysteme. Seit 1989 werden in den eigenen Schulungsräumen diverse Kurse abgehalten, unter anderem auch C-Kurse für Sprachumsteiger und für den MS C-Compiler 5.1.

Word 5.0 jetzt für MS-DOS und MS OS/2 :

Dieses ist der fünfte Streich ...

... und der sechste, der kann ruhig noch ein wenig warten, denn mit der Version 5.0 – die ab September im Handel ist – hat Word wieder einen mächtigen Sprung nach vorn gemacht. Microsoft unterstreicht damit ihren Führungsanspruch in Sachen Textverarbeitung in der Bundesrepublik, wo man nach Angaben des unabhängigen Marktforschungsinstituts Dataquest mit Word bereits 1988 das hierzulande meistverkaufte Softwarepaket überhaupt anbot.

Da wir davon ausgehen, daß Sie sich als Leser des Microsoft System Journals bereits mehr oder weniger gut mit Microsoft Word auskennen, wollen wir in diesem Artikel nur die neuen Features von Word 5.0 kurz vorstellen. Daran anschließend gehen wir dann auf die neuen Desktop Publishing-Fähigkeiten und die verbesserten Möglichkeiten der Makrosprache etwas intensiver ein.

Die neuen Features

Wie sein Vorgänger, setzt das neue Microsoft Word 5.0 erneut umfangreiche und zukunftsweisende Standards für den Markt der professionellen Textverarbeitung: Eine Vielzahl neuer und stark praxisorientierter Funktionen, die weit über die herkömmlicher Texterfassung hinausgehen, macht diese Version, die sowohl unter MS-DOS wie auch unter MS OS/2 lauffähig ist, noch vielseitiger und bedienerfreundlicher. Die wesentlichen neuen Features, die das noch schneller gewordene Microsoft Word 5.0 zu einem High-End-Textverarbeitungsprogramm machen, sind:

- Seitenumbruch im Hintergrund
- Darstellung von Textspalten nebeneinander
- Integration von Grafik in vielen Formaten
- absolute Positionierung von Objekten wie Absätzen oder Bildern auf einer Seite
- Layoutkontrolle in WYSIWYG-Darstellung
- erweiterte Rechtschreibprüfung
- deutscher Thesaurus
- frei wählbare automatische Sicherung
- Auswahl von Laufwerken und Verzeichnissen am Bildschirm
- Erweiterung der Menü-Zusätze auch mit Fenster-Optionen
- Setzen von Textmarken
- erweiterte Querverweise
- verbesserter Dateimanager
- erweiterte Makro-Sprache
- vereinfachtes Setzen der Tabulatoren
- einfache Konvertierung in andere Textformate
- flexible *Übertragen*-Optionen
- beidseitiges Drucken und Farbdruck
- Druckfarbe für Zeichenformatierung
- installierte Bildschirmtreiber
- Nutzung von Expanded Memory
- MS OS/2-Unterstützung

Doch lassen Sie uns die neuen Eigenschaften von Word etwas genauer betrachten.

Rechtschreibprüfung jetzt voll integriert

Die erhöhte »Intelligenz« von Microsoft Word 5.0 zeigt sich auch im Umgang mit der deutschen Sprache: Das Programm verfügt über ein integriertes und sehr bedienerfreundliches Rechtschreibprogramm, das 130.000 Wörter kennt und für das nun keine extra Arbeitsdatei mehr erstellt werden muß.

```
AS   Sobald Sie den Befehl Bibliothek
     Rechtschreibung gewählt haben, beginnt Word 5,
     Ihren Text zu überprüfen: es muß also kein
     separater Befehl Prüfen gewählt werden, wie das
     bei früheren Versionen der Fall war. Der
     Rechtschreibungs-Ausschnitt nimmt nicht mehr
     den ganzen Bildschrim ein. Der Text-Ausschnitt
     ist teilweise sichtbar, damit unbekannte Wörter
     in ihrem inhaltlichen Zusammenhang angezeigt
     werden können.¶
AS   Die Korrekturen werden jetzt sofort im Text
     vorgenommen und nicht erst beim Verlassen des
     Rechtschreibprogramms. Außerdem dürfen sie

Bildschirm                    Bildschirme
Bildschirms                   Bildschirmen
Bildschirmes                  Bildschnitt

Bildschrim nicht gefunden

RECHTSCHREIBUNG: Korrektur Aufnehmen Quitt Ignorieren Optionen Rückgängig

Korrekturauswahl für unbekanntes Wort, dann EINGABETASTE oder unterbrechen Sie!
Se2 Sp26        (I)        ?                                    WORDS.TXT
```

◂◂ *Bild 1: Die Rechtschreibhilfe wird jetzt nicht mehr separat geladen, sondern ist voll in Word integriert.*

Sobald Sie den Befehl *Bibliothek Rechtschreibung* gewählt haben, beginnt Word 5, Ihren Text zu überprüfen; es muß also kein separater Befehl *Prüfen* gewählt werden, wie das bei früheren Versionen der Fall war.

Der Rechtschreibungs-Ausschnitt nimmt auch nicht mehr den ganzen Bildschirm ein. Der Text-Ausschnitt ist teilweise sichtbar, damit unbekannte Wörter in ihrem inhaltlichen Zusammenhang angezeigt werden können.

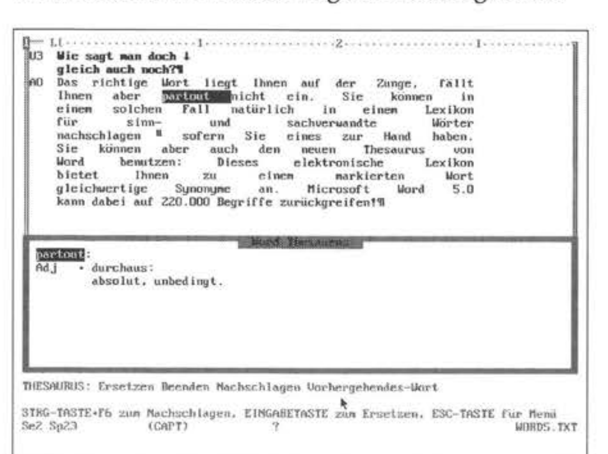
Die Korrekturen werden jetzt sofort im Text vorgenommen und nicht erst beim Verlassen des Rechtschreibprogramms. Außerdem dürfen sie auch Leerzeichen und Interpunktionszeichen enthalten.

Eine weitere wichtige Verbesserung: Sie können nun die Befehle des total neu gestalteten Menüs *Bibliothek Rechtschreibung* auch in Makros verwenden.

Wie sagt man doch gleich auch noch?

Ein nicht seltener Fall: Das richtige Wort liegt Ihnen auf der Zunge, fällt Ihnen aber partout nicht ein. Sie können in einem solchen Fall natürlich in einem Lexikon für sinn- und sachverwandte Wörter nachschlagen – sofern Sie eines zur Hand haben.

Sie können aber auch den neuen Thesaurus von Word benutzen: Dieses elektronische Lexikon bietet Ihnen zu einem markierten Wort gleichwertige Synonyme an. Microsoft Word 5.0 kann dabei auf 220.000 Begriffe zurückgreifen!



◂◂ *Bild 2: Sollte Ihnen ein Wort nicht gefallen, wählen Sie mit dem Thesaurus einfach ein anderes aus.*

Betriebssystem- und Netzwerk-Unterstützung wurden verbessert

Microsoft Word 5.0 unterstützt die Betriebssysteme MS-DOS *und* MS OS/2. Dabei stellt die neue Version selbständig fest, ob sie unter MS-DOS, MS OS/2 bzw. dem MS LAN-Manager gestartet wurde und konfiguriert sich entsprechend von selbst. Ein Vorteil, der besonders bei der Installation im Netz zum Tragen kommt, da Word 5.0 von ein und demselben Server gestartet werden und sowohl auf MS-DOS- als auch MS OS/2-Workstations laufen kann. Damit ist Microsoft Word 5.0 prädestiniert auch für den Einsatz in gemischten Netzen.

Bisher war eine spezielle Netzwerk-Version erforderlich, um mehreren Workstations den Zugriff auf Word zu ermöglichen. Word 5 dagegen kann jetzt standardmäßig sowohl in einem Einzelbenutzersystem als auch in einem Netzwerk laufen.

Zur Installation in einem Netzwerk wird Word einfach auf einem Server eingerichtet; anschließend müssen dann nur noch die einzelnen Arbeitsstationen für Word eingerichtet werden. Um Word auf dem Netzwerk-Server einzurichten, führen Sie das Setup-Programm aus und wählen die Option »Netzwerk«; zum Einrichten der Arbeitsstationen führen Sie auf jeder eine spezielle Version von dem Setup-Programm aus.

An Ihrem Arbeitsplatz-PC können Sie mit ihren eigenen Textbausteindateien, Druckformatvorlagen und Wörterbüchern arbeiten und mit dem Befehl *Zusätze* oder mit anderen Befehlen selbst Standardeinstellungen festlegen.

Denken Sie daran, daß Sie gemäß Lizenzvertrag für jeden Netzwerkbenutzer, der mit Microsoft Word arbeitet, ein Handbuch und eine Lizenz erwerben müssen. Diese sind in Form von Microsoft Word Arbeitsstations-Pakets erhältlich.

Querverweise werden automatisch aktualisiert

Wenn Sie oft Querverweise in Ihre Texte einfügen, wird sich die neue Querverweisfunktion von Word als äußerst zeitsparend erweisen. Im Moment gehen Sie wahrscheinlich folgendermaßen vor:

Während des Erstellens und Überarbeitens eines Textes geben Sie den Querverweistext vermutlich so ein: »Sehen Sie dazu auf Seite x nach«, oder »Siehe Abbildung x«. Sie geben den Buchstaben »x« anstelle der Seitenzahl oder Abbildungsnummer ein, weil Ihnen die betreffende Nummer erst nach dem endgültigen Seitenumbruch bekannt sein wird. Wenn der dann festliegt, bestimmen Sie die korrekten Querverweisnummern und fügen sie anstelle der »x« in Ihren Text ein – und hoffen, daß sich der Umbruch dadurch nicht mehr verschiebt.

In Word 5 können Sie Textabschnitte, auf die Sie verweisen möchten, auch speziell markieren. Wenn Sie einen Querverweis erstellen möchten,

geben Sie an der betreffenden Stelle einen speziellen Code ein. Soweit gleicht dies dem oben erwähnten Verfahren, in dem die Stelle, an der später eine Nummer eingefügt wird, mit einem »x« markiert wird. Wenn Sie jedoch diesen speziellen Code verwenden, fügt Word beim Drukken des Texts die korrekten Nummern automatisch ein.

Das Verfahren sieht etwa folgendermaßen aus: Sie kennzeichnen mit *Format tExtmarke* z.B. den Text auf der Seite, auf die Sie hinweisen wollen. An der Stelle, an der der Querverweis erscheinen soll, geben Sie `Seite:`, dann den Namen der Textmarke ein und und drücken F3. Dadurch machen Sie aus dem Querverweis einen Textbaustein, ähnlich den vorgegebenen wie Datum oder Seite. Der Abschnitt könnte dann so aussehen:

`Sehen Sie dazu auf Seite (Seite:Text) nach.`

Mit dieser Funktion können Sie automatische Querverweise auf Seitenzahlen, Absatz- und Fußnotennummern erstellen. Sie können auch selbst eine Serie von Elementen (zum Beispiel Tabellen oder Illustrationen) festlegen und die einzelnen Elemente von Word automatisch numerieren lassen. Wenn Sie dann einen Querverweis auf eines dieser Elemente erstellen, wird Word die betreffende Nummer automatisch einfügen.

Nichts geht mehr ...
... verloren

Die neue automatische Speicherfunktion bietet Ihnen die Möglichkeit, während der Arbeit automatische Sicherungskopien Ihrer Dateien erstellen zu lassen. Sie geben den gewünschten Zeitabstand ein und Word wird automatisch alle Änderungen speichern, die Sie seit dem letzten Speichervorgang (sei er nun automatisch oder mit dem Befehl *Übertragen Speichern* eingeleitet worden) vorgenommen haben. Die Sicherungskopien, die mit der Funktion Auto-speichern erstellt worden sind, beinhalten alle Texte, alle Druckformatvorlagen und die aktuelle Textbausteindatei.

```
AUSSCHNITT ZUSÄTZE Ausschnitt Nr.: 1
     Verborgener Text sichtbar:(Ja)Nein              Zeilenlineal:(Ja)Nein
        Sonderzeichen sichtbar: Nein(Teilweise)Alle        Layout:(Ja)Nein
               Zeilenumbrüche:(Ja)Nein                  Gliederung: Ja(Nein)
            Druckformatspalte:(Ja)Nein
ALLGEMEINE ZUSÄTZE Warnton aus: Ja(Nein)          Kurzinformation: Ja(Nein)
                    Maßeinheit:(Zoll)Cm 10er-Teilung 12er-Teilung Punkt
                    Bildschirm: 5                  Seitenumbruch:(Auto)Manuell
                        Farben:                     Auto-speichern: 10
Auto-speichern mit Bestätigung:(Ja)Nein              Menü sichtbar:(Ja)Nein
            Ausschnittsrahmen:(Ja)Nein       Dezimaltrennzeichen: .(,)
                    Zeitformat: 12(24)          Abstand Tabstopps: 0,49"
                 Zeilennummern: Ja(Nein)        Leerzeilen zählen: Ja(Nein)
              Geschwindigkeit: 3                    Linienzeichen: (|)
              Rechtschreibung: C:\WORD5\SPELL-GE.LEX
Geben Sie bitte das Intervall für Autospeichern in Minuten an, 0 für nicht!
```

▸ *Bild 3: Im wesentlich erweiterten Menü »Zusätze« kann die automatisch Speicherung mit Intervall und Abfrage eingestellt werden.*

Sie können Word auch anweisen, vor jedem Speichervorgang eine entsprechende Bestätigung auszuführen. Wenn dann bei der Arbeit plötzlich nichts mehr geht, sehen Sie sich die Meldungszeile an. Dort werden Sie von Word gefragt, ob Sie wirklich speichern wollen. Auf diese Weise könne Sie verhindern, daß Bearbeitungen, die Sie vielleicht gar nicht übernehmen wollten, automatisch gesichert werden.

Sollte es während der Arbeit zu einem Absturz kommen, sind die Dateien wiederherstellbar. Beim erneuten Starten stellt Word automatisch fest, ob Dateien wiederhergestellt werden müssen und macht das dann auf Wunsch für Sie.

Schnelleres und leichteres Setzen der Tabstopps

Die Tabstopps lassen sich jetzt sowohl mit Hilfe des Befehls *Format Tabulator* als auch mit der Maus wesentlich einfacher setzen. So können Sie die Markierung auf dem Zeilenlineal auf die gewünschte Position bringen und einen Tabstopp setzen, indem Sie einfach den ersten Buchstaben der gewünschten Ausrichtung eingeben: L für links, Z für zentriert, R für rechts, D für dezimal oder V für eine vertikale Linie. Sie können die gewünschte Ausrichtung natürlich auch im Befehlsfeld *Ausrichtung* des Befehls *Format Tabulator Setzen* (die einzig mögliche Methode in Word 4) vornehmen.

Wenn Sie sich das Zeilenlineal anzeigen lassen, können Sie die Tapstopps bequem mit der Maus setzen, verschieben oder löschen, indem Sie einfach auf die betreffende Stelle klicken bzw. den Mauszeiger auf dem Zeilenlineal ziehen. Sie müssen den Befehl *Format Tabulator Setzen* also überhaupt nicht mehr wählen. Sie können sogar Absatzeinzüge ändern, indem Sie auf das Einzugssymbol klicken und es über das Zeilenlineal ziehen.

```
[-- RL·················1···············Z·····R·········3·················4·]
     Wenn   Sie   sich   das   Zeilenlineal  anzeigen
    lassen,  können  Sie  die  Tapstopps  bequem  mit  der
    Maus   setzen,   verschieben   oder   löschen,   indem
    Sie  einfach  auf  die  betreffende  Stelle  klicken
```

▸▸ *Bild 4: Hier wurde gerade ein rechter Tab mit dem Mauszeiger eingerichtet.*

Das Zeilenlineal erleichtert Ihnen auch das Erstellen und Überprüfen von Tabellen, denn die Einteilung des Lineals wird automatisch an die proportionalen Schriftarten angepaßt (*Bild 4*). Auf dem Bildschirm können proportionale Schriftarten nicht als solche angezeigt werden; wenn Sie also Schriftarten wie zum Beispiel Times-Roman und Helvetica verwenden, ändert Word die Markierungen auf dem Zeilenlineal automatisch so, daß die Tabellen auf dem Bildschirm genauso ausgerichtet werden wie im gedruckten Text.

Mit F1 geht's in alle Verzeichnisse

Wenn Sie eine Datei von der Diskette oder Festplatte laden möchten, können Sie nun die Taste [F1] drücken, um Dateien von einem beliebigen Verzeichnis oder Unterverzeichnis anzeigen zu lassen und auszuwählen. Dabei kann es sich um Textdateien, eine Textbausteindatei, einen Druckertreiber oder beliebige andere Dateien handeln. Damit sind Sie in der Lage, eine Datei aus einem anderen Laufwerk oder Verzeichnis zu laden, ohne den gesamten Suchweg eingeben oder die Angaben im Befehlsfeld *Laufwerk/Verzeichnis:* des Befehls *Übertragen Optionen* ändern zu müssen.

```
C:\WORD5\*.TXT
CAPTURE.TXT      MACROCNV.TXT     WORD_RTF.TXT     [C:]
CHARTER.TXT      MAKEVID.TXT      ZMTEST.TXT       [D:]
GRAFIK.TXT       OSZ_INFO.TXT     [..]             [E:]
INFO.TXT         WORD_DCA.TXT     [A:]             [F:]
JARGON.TXT
```

Die Liste, die angezeigt wird, wenn Sie die Taste F1 drücken, enthält sämtliche Dateien im aktuellen Verzeichnis, die Namen aller eventuellen Unterverzeichnisse des aktuellen Verzeichnisses, die Laufwerksbezeichnungen Ihres Systems sowie eine Bezeichnung für das übergeordnete Verzeichnis des aktuellen Verzeichnisses.

Laufwerks- und Verzeichnisnamen sind in eckigen Klammern eingeschlossen, um sie von ladbaren Dateien zu unterschieden; so zeigt zum Beispiel `[A:]` Laufwerk A: an, `[BERICHTE]` weist auf ein Unterverzeichnis mit dem Namen `BERICHTE` hin, und das Symbol `[..]` deutet auf ein übergeordnetes Verzeichnis hin. Wenn Dateien aus einem anderen Laufwerk oder Verzeichnis aufgelistet werden sollen, wählen Sie einfach die betreffende Bezeichnung aus der Liste und drükken nochmals die Taste F1 – und wenn Sie dabei eine Liste in Laufwerk A anzeigen lassen wollen, ohne eine Diskette eingelegt zu haben, nimmt Word Ihnen das auch nicht übel.

Verknüpfen von Texten mit Tabellen, Grafiken und sogar anderen Texten

Ähnlich wie beim Verknüpfen von Kalkulationstabellen oder Grafiken können, Sie auch den Text, an dem Sie gerade arbeiten, mit einem anderen Word-Text oder mit einem Teil davon verknüpfen.

```
BIBLIOTHEK VERKNÜPFEN DOKUMENT Dateiname: E:\MSJ35\WORD5\WORD5.TXT
                                Textmarke: zentrierter Absatz
Geben Sie bitte den Namen der Textmarke ein oder wählen Sie einen mit F1!
```

Nehmen wir an, Sie haben eine Hauptpreisliste aller Ihrer Produkte und möchten für bestimmte Produkte separate Broschüren erstellen. Mit Word können Sie eine bestimmte Preisgruppe in der Hauptpreisliste markieren und den betreffenden Abschnitt mit dem Text einer Broschüre verbinden. Nach jeder Preisänderung, die in der Hauptpreisliste vorgenommen wird, können Sie dann mit der automatischen Aktualisierungs-Funktion von Word die gleichen Änderungen auch in der Broschüre vornehmen lassen.

Zum Markieren des zu verknüpfenden Textabschnitts verwenden Sie die neue Funktion *Textmarke* von Word; anschließend wird der Textabschnitt mit Hilfe des Befehls *Bibliothek verKnüpfen Dokument* mit dem aktuellen Text verknüpft.

Komfortable Bildschirmanzeige mit acht Betriebsarten

Im Befehlsfeld *Bildschirm:* des Befehls *Zusätze* können Sie jede beliebige Text- oder Grafik-Betriebsart wählen, die von Ihrem Bildschirm-Adapter unterstützt wird.

```
                    H  L M N P  = (Abschalten)
Hintergrund: Ausschnitt 1:              Ausschnittsrahmen: Mustertext
            Menüs: Mustertext                   Meldungen: Mustertext
      Optionsmenü: Mustertext                 Statuszeile: Mustertext
 8,5 Punkte oder weniger:                  Großbuchstaben: Mustertext
  9,0 bis 10 Punkte: Mustertext                Kapitälchen: Mustertext
 10,5 bis 12 Punkte: Mustertext               Tiefgestellt: Mustertext
 12,5 bis 14 Punkte: Mustertext               Hochgestellt: Mustertext
 Mehr als 14 Punkte: Mustertext           Verborgener Text: Mustertext
              Fett: Mustertext             Durchgestrichen: Mustertext
            Kursiv: Mustertext            Fett und Kursiv: Mustertext
     Unterstrichen: Mustertext    Fett und Unterstrichen: Mustertext
Doppelt unterstrichen: Mustertext Kursiv und Unterstrichen: Mustertext

  Verborgener Text sichtbar:(Ja)Nein          Zeilenlineal:(Ja)Nein
    Sonderzeichen sichtbar: Nein(Teilweise)Alle     Layout: Ja(Nein)
          Zeilenumbrüche:(Ja)Nein                Gliederung: Ja(Nein)
        Druckformatspalte:(Ja)Nein
ALLGEMEINE ZUSÄTZE Warnton aus: Ja(Nein)   Kurzinformation: Ja(Nein)
            Maßeinheit:(Zoll)Cm 10er-Teilung 12er-Teilung Punkt
            Bildschirm: 1                Seitenumbruch:(Auto)Manuell
               Farben:                      Auto-speichern: 10
Auto-speichern mit Bestätigung:(Ja)Nein       Menü sichtbar:(Ja)Nein
        Ausschnittsrahmen:(Ja)Nein    Dezimaltrennzeichen: .(,)
            Zeitformat: 12(24)            Abstand Tabstopps: 0,49"
          Zeilennummern: Ja(Nein)       Leerzeilen zählen: Ja(Nein)
        Geschwindigkeit: 3                  Linienzeichen: (|)
        Rechtschreibung: C:\WORD5\SPELL-GE.LEX
F1 zur Elementauswahl. Buchstaben oder BILD NACH OBEN/UNTEN-TASTE für Farbe!
```

In Word 4 waren nur die Betriebsarten *Text* und *Grafik* möglich. Jetzt sind es auf einem EGA-Monitor drei Textbildschirme mit 25, 43 und 50 Zeilen sowie fünf Grafikdarstellungen mit 25, 30, 34, 43 und 60 Zeilen. Word 5 merkt sich dabei die aktuelle und die vorherige Einstellung im Befehlsfeld *Bildschirm:*, und Sie können mit der Tastenkombination Alt F9 zwischen diesen beiden Betriebsarten umschalten.

Druckerunterstützung erweitert und verbessert

Der endgültige Prüfstein eines Textverarbeitungsprogramms liegt in der Qualität des gedruckten Textes. Word bietet seit der ersten Version eine ausgefeilte Druckerunterstützung; dadurch war es Ihnen möglich, die Leistungsfähigkeit Ihres Druckers voll auszunutzen und höchste Druckqualität zu erzielen. In Word 5 wurden mehr als 50 neue Druckermodelle berücksichtigt. Außerdem wurde das Handbuch mit den Druckerinformationen gründlich überarbeitet und aktualisiert.

Im Befehl *Druck Option* können Sie nun auch außer der Druckertreiberdatei (DBS) Ihr spezielles Druckermodell angeben. Wenn Sie zum Beispiel einen HP Laserjet II haben, könnten Sie im Befehlsfeld *Drucker:* `HPLASMS` und im Befehlsfeld *Modell:* `Laserjet Serie II` wählen. Wenn Sie in den beiden Befehlsfeldern jeweils die Taste F1 drücken, wird eine Liste der verfügbaren Optionen angezeigt.

Bei Laserdruckern berücksichtigt Word jetzt auch die Papierbereiche, die nicht bedruckt werden. Sie können somit die exakte Papiergröße und genauen Seitenränder angeben, ohne sich um den tatsächlichen Druckbereich kümmern zu müssen. Für gewisse Laserdrucker werden auch der manuelle Papiervorschub und der Vorschub von Briefumschlägen unterstützt.

Sollten Sie mit einem PostScript-Drucker arbeiten, können Sie alle geradzahligen Schriftarten von bis zu 126 Punkten verwenden, die von Ihrem Drucker unterstützt werden.

Wenn Sie einen Drucker haben, der beidseitig drucken kann (wie zum Beispiel der HP Laserjet 2000, Canon LBP-811R oder Toshiba Page Laser), geht auch das im neuen Befehlsfeld *Beidseitig:* einzustellen. Haben Sie einen HP Laserjet

◄◄ *Bild 5: Die Auswahl der Dateien beschränkt sich nicht mehr nur auf das eingestellte Verzeichnis.*

◄ *Bild 7: Wie Sie sehen, können Sie einen Farbmonitor im Textmodus beliebig bunt gestalten.*

◄◄ *Bild 6: Um Textdateien zu verknüpfen, können Sie eine Textdatei und eine Textmarke angeben bzw. mit F1 auswählen.*

2000, Laserjet II D, Mannesmann Tally 910, Panasonic Laser 4450, Canon L8P-811-R, Toshiba Page Laser oder einen PostScript-Drucker, dann können Sie im gleichen Text sowohl Seiten im Hochformat als auch Seiten im Querformat drucken.

Wenn Sie mit einem Farbdrucker arbeiten, können Sie dem ausgewählten Textabschnitt im Befehlsfeld *Farbe:* des Befehls *Format Zeichen* eine Farbe zuweisen, genauso, wie Sie die Zeichen fett oder kursiv formatieren würden. Sie können auch Rahmenlinien- und Hintergrundschattierungsfarben bestimmen.

```
UBERTRAGEN OPTIONEN Laufwerk/Verzeichnis: E:\MSJ35\WORD5
            Speichern zwischen Sitzungen: Ja Nein
Wählen Sie bitte eine Option!
```

► *Bild 8: Word merkt sich auf Wunsch das Verzeichnis, in dem gearbeitet wird.*

Neues im Menü »Übertragen Optionen«

Schluß mit dem Ärger, daß das zuletzt eingestellt Verzeichnis nach dem Verlassen von Word verlorengeht. Das Laufwerk und Verzeichnis, das Sie im Menü *Übertragen Optionen* für das Laden und Speichern von Dateien festlegen, kann auf Wunsch als Standardeinstellung bestimmt werden, die zu Beginn jeder Word-Arbeitssitzung automatisch gültig ist.

Word als echtes Desktop Publishing-Werkzeug

Welche Möglichkeiten zum Desktop Publishing Word 5 nun hat, sehen Sie am besten am neuen *Microsoft System Journal*: Die Ausgabe wurde komplett mit Word 5 gestaltet, auf einem PostScript-Drucker zur Korrektur ausgedruckt und die Filme auf einer Linotronic 300 belichtet.

Die wohl mit interessantesten Innovationen von Word 5 – neben den Makro-Neuheiten – sind die verschiedenen Funktionen zur Gestaltung des Seitenlayouts, mit denen Sie raffiniert ausgelegte Broschüren, Berichte, Werbematerial, illustrierte Bücher und ähnliches erstellen können. Die neuen Publishing-Funktionen von Word bieten Ihnen folgende Möglichkeiten:

- Importieren, Festlegen der Größe und Drucken von Grafiken, die mit verschiedenen Programmen erstellt wurden.
- Positionieren eines beliebigen Absatzes, der einen Text oder eine Grafik enthalten kann, an einer bestimmten, fixierten Stelle auf der Seite. Der bestehende Text wird automatisch um den positionierten Absatz herum ausgedruckt.
- In der Betriebsart *Layout* werden Spalten nebeneinander auf dem Bildschirm angezeigt; positionierte Textabschnitte und Grafiken werden durch einen entsprechenden Rahmen dargestellt.
- Überprüfen des Seitenlayouts vor dem Drukken. In der Betriebsart *Druck Layoutkontrolle* zeigt Word wahlweise eine oder zwei Seiten so auf einem Grafikbildschirm an, wie sie gedruckt aussehen werden. Dabei sind alle einzelnen Elemente erkennbar, einschließlich der Grafiken, positionierten Absätze und Kopf-/Fußzeilen.
- Einfaches Kombinieren von unterschiedlichen Spaltenformaten auf der gleichen Seite.
- Aktivieren der Option *Zeilenumbrüche*, damit Word die Einteilungen auf dem Zeilenlineal den Proportionalschriftarten anpaßt.
- Hinzufügen von Hintergrundschattierungen einer bestimmten Intensität.

Vereinfachte Grafik-Integration

Sie können nun Grafiken auf dieselbe Weise importieren wie Daten aus Kalkulationstabellen, d.h. mit dem Befehl *Bibliothek verKnüpfen*. Dabei kann jede Grafik beliebig vergrößert oder verkleinert werden – und das mit den original Proportionen oder verzerrt. Microsoft Word 5 unterstützt dabei folgende Grafikformate: Windows Clipboard Bitmapped, PC-Paintbrush PCX-Format, Pageview Bitmapped, Postscript-EPS-Dateien und -Druckdateien, Lotus PIC-Grafiken, HPGL- und TIFF-Dateien.



►► *Bild 9: Die Grafiken können ebenso ausgewählt werden wie Dateien – es sei denn, Sie wollen den Punktbefehl direkt eingeben.*

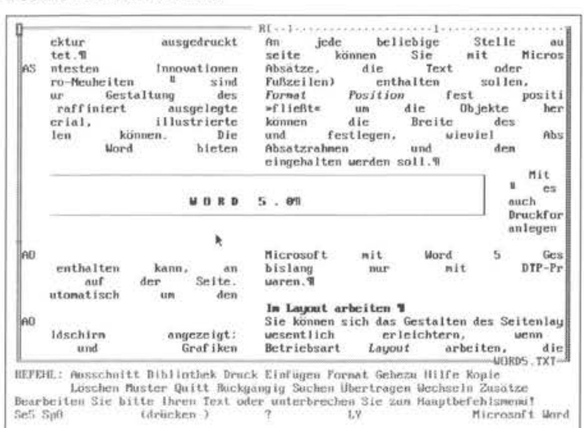
Außerdem wird zu der neuen Word-Version das Capture-Programm mitgeliefert, mit dem Sie vom Bildschirm – aus beliebigen Programmen heraus – Word-kompatible Grafiken erzeugen können. Als Beispiele für die Arbeit von Capture sehen Sie die Bilder dieses Artikels und viele andere Bilder dieser Ausgabe.

Lassen Sie den Text um Objekte fließen

An jede beliebige Stelle auf einer Dokumentenseite können Sie mit Microsoft Word 5 nun Absätze, die Text oder Grafik (auch Kopf-/ Fußzeilen) enthalten sollen, über den Befehl *Format Position* fest positionieren. Der Text »fließt« um die Objekte herum (*Bild 12*). Sie können die Breite des Absatzrahmens bestimmen und festlegen, wieviel Abstand zwischen dem Absatzrahmen und dem benachbarten Text eingehalten werden soll.

Mit dieser Möglichkeit – es lassen sich dafür auch Muster-Druckformatvorlagen anlegen – schafft Microsoft mit Word 5 Gestaltungsvaria-

tionen, die bislang nur mit DTP-Programmen realisierbar waren.



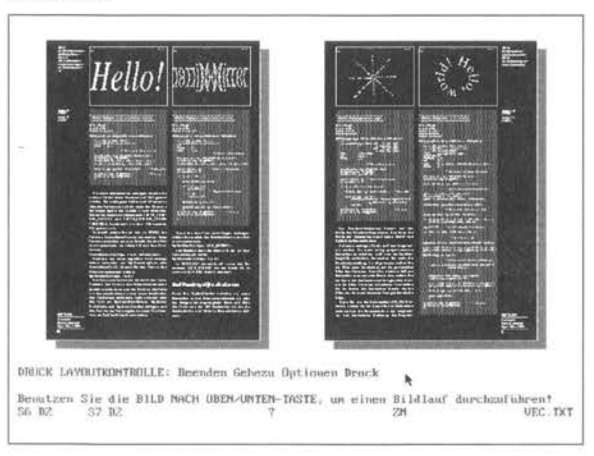
◂◂ *Bild 10: Der zentrierte Schriftzug »Word 5.0« ist hier in der Betriebsart Layout dargestellt.*

Im Layout arbeiten

Sie können sich das Gestalten des Seitenlayouts wesentlich erleichtern, wenn Sie in der Betriebsart *Layout* arbeiten, die Sie entweder mit dem Befehl *Zusätze* oder durch Drücken der Tastenkombination [Alt][F4] aktivieren. In dieser Betriebsart zeigt Word nicht nur Spalten bzw. Absätze nebeneinander an, sondern auch die Rahmen der Grafik- oder Textabsätze, die mit dem Befehl *Format Position* positioniert wurden. Die eigentlichen Grafiken werden jedoch nicht angezeigt.

Sie können somit am Monitor die Druckbilddarstellung kontrollieren und gegebenenfalls ändern, ohne den Weg über den Ausdruck gehen zu müssen.

Einen Spaltenumbruch können Sie jetzt z.B. ganz einfach einfügen (d.h. wenn Sie eine Spalte beenden und oben in der nächsten Spalte fortfahren möchten), indem Sie [Ctrl][Alt][Return] drücken.



◂◂ *Bild 11: Erkennen Sie die Seiten dieser Abbildung? Sie wurde mit dem Capture-Programm von einer Layoutdarstellung dieser Ausgabe gemacht.*

Die Kontrolle des Layouts

Microsoft Word 5 enthält nun auch eine Layoutkontrolle, mit der am Monitor die Ganzseitenansicht des Dokuments ein- oder zweiseitig möglich ist.

Damit können auf dem Bildschirm die einzelnen Dokumentseiten genau so dargestellt werden, wie sie auf dem Drucker ausgegeben werden sollen – Stichwort WYSIWYG. Dabei werden auch alle Grafiken (außer Druckdateien einschließlich Postscript) sowie die Farben der Grafiken und Zeichen sichtbar, sofern ein entsprechender Farbdruckertreiber geladen wurde.

Kombinieren von unterschiedlichen Spaltenformaten auf der gleichen Seite

Mit den bisherigen Word-Versionen mußte man etwas tricksen, um z.B. eine einspaltige Überschrift über einen zweispaltigen Text zu bringen – und das funktionierte auch nur, wenn die Überschrift am Seitenanfang und nicht irgendwo in der Mitte stehen konnte.

Jetzt geben Sie einfach einen Bereichswechsel ([Ctrl][Return]) ein und wählen anschließend im Befehlsfeld *Bereichswechsel:* des Befehls *Format Bereich Layout* die Option *Fortlaufend*, wenn Sie die Spaltenformatierung einer Seite ändern, d.h. von einem einspaltigen auf ein dreispaltiges Layout wechseln möchten.

So sehen Sie den Zeilenfall

Wenn Sie Proportionalschriftarten verwenden und im Befehlsfeld *Zeilenumbrüche:* des Befehls *Zusätze* die Option *Ja* gewählt haben oder einfach die Tastenkombination Alt F7 betätigen, wird die Einteilung auf dem Zeilenlineal der verwendeten Schriftart und dem Schriftgrad angepaßt, da die eigentlichen Proportionalschriftarten nicht als solche auf dem Bildschirm angezeigt werden können.

WORD 5.0

◂◂ *Bild 12: Dieser Absatz wurde einfach zwischen den Seitenrändern zentriert.*

Somit können Sie sich das Erscheinungsbild Ihres gedruckten Textes besser vorstellen, was besonders bei Tabellen von Vorteil ist. Wenn Sie in der Betriebsart *Layout* arbeiten, wählt Word automatisch diese Betriebsart.

Schattieren Sie hervorzuhebende Absätze

Mit dem Befehl *Format Rahmen* können nun den Absätzen Hintergrundschattierungen hinzugefügt werden (vorausgesetzt, Ihr Drucker unterstützt die Schattierung). Geben Sie dazu im Befehlsfeld *Hintergrundschattierung:* einfach eine entsprechende Prozentzahl zwischen 0 (keine Schattierung) und 100 (schwarz) ein.

Dafür gibt es in diesem Heft mit den hinterlegten Listings ebenso ein Menge Beispiele wie für die Möglichkeit, Seiten zu spiegeln – so geschehen mit der Marginalspalte für Bildunterschriften und die Fußzeilen. Wenn Sie im Befehlsfeld *Ränder spiegeln:* des Befehls *Format Bereich Seitenrand* die Option *Ja* wählen, werden gegenüberliegende Seiten identische innere und äußere Seitenränder aufweisen.

Leistungsumfang der Makros wesentlich angereichert

An den Makrofunktionen von Word 5 sind zahlreiche Verbesserungen vorgenommen worden, besonders im Bereich der Sonderanweisungen, die beim Schreiben von Makros eingefügt werden können.

An dieser Stelle sei auch gleich ein Hinweis angebracht: Da einige Word-Menüs geändert wurden, müssen die mit Word 4 erstellten Makros konvertiert werden, bevor Sie sie in der Version 5 ausführen. Dazu gibt es auf der Hilfsprogrammdiskette das Programm MACROCNV.EXE mit den Erläuterungen zur Handhabung in der Textdatei MACROCNV.TXT.

Doch kommen wir nun zu den Erweiterungen der Makrosprache.

Neue reservierte Variablen.
Mehrere neue reservierte Variablen erschließen ungeahnte Möglichkeiten für die Makro-Anwendung:
Ausschnitt zum Bestimmen oder Ausfindigmachen der Nummer des aktuellen Ausschnitts.

Mit der Varaible *Ausschnitt* können Sie die Nummer des Ausschnitts feststellen, in dem Sie sich gerade befinden.

```
«AWENN Ausschnitt = "3"»
```

Sie können aber auch in einen bestimmten Ausschnitt wechseln:

```
«BESTIMMEN Ausschnitt = "3"»
```

Echo zum Steuern der Bildschirm-Aktualisierung während der Makro-Ausführung.

Mit der Variablen *Echo* läßt sich die Anzeige der Bildschirmaktivitäten ausschalten und damit eine höhere Arbeitsgeschwindigkeit erreichen. Die beiden zulässigen Werte sind *an* und *aus.*

```
«BESTIMMEN Echo = "an"»
«BESTIMMEN Echo = "aus"»
```

Bevor Sie den Wert ändern, können Sie ihn natürlich auch testen:

```
«AWENN Echo = "an"»
```

Eingabemodus für die leichtere Handhabung von Word-Eingabeaufforderung während der Makro-Ausführung.

Mit der Variable *Eingabemodus* können Sie festlegen, ob die Antworten auf eventuelle Word-Eingabeaufforderungen vom Makro oder vom Anwender des Makros gegeben werden sollen, oder ob sie gar zu ignorieren sind. Wenn Sie z.B. im Makro die Speicherfunktion ausführen, kann es sein, daß Word einen Sicherheitsabfrage macht (J um Änderungen im Dokument zu speichern N wenn nicht oder ...). Sie haben eine der drei folgenden Möglichkeiten, diese Abfrage abzufangen:

```
«BESTIMMEN Eingabemodus = "Benutzer"»
«BESTIMMEN Eingabemodus = "Makro"»
«BESTIMMEN Eingabemodus = "abschalten"»
```

Word nimmt standardmäßig den Wert *Makro* an. Das Makro muß dann also auf solche Eventualitäten vorbereitet sein.
Speichern zum Wahrnehmen der Meldung SPEICHERN, mit der Word zum Sichern der Arbeit auffordert.

Mit dieser Variablen unterstützt Word alle diejenigen, die umfangreiche Änderungen an ihren Texten vornehmen und kein Extended Memory besitzen. Dieser Boolsche Operator meldet entweder »richtig« oder »falsch«. Er sollte bei Wechsel- und Sortiervorgängen, beim Erstellen von Inhalts- und Stichwortverzeichnissen sowie bei umfangreichen Bearbeitungen und komplexen mathematischen Berechnungen folgendermaßen eingesetzt werden:

```
«AWENN Speichern»
        <Unt>ÜA
«EWENN»
```

Wordversion zum Feststellen der Versionsnummer des Word-Programms (vorgesehen hauptsächlich für zukünftige Versionen).

Neue Funktionen, logische Operatoren und Matrizen
Einige neue Funktionen, logische Operatoren und Matrizen erleichtern ebenfalls den Umgang mit dem Programm.
LÄNGE (auch als *LEN* bekannt) mißt die Länge einer Zeichenfolge in Anzahl der Zeichen.

Mit dieser Funktion können Sie die Länge einer Konstanten oder Variablen feststellen:

```
«BESTIMMEN Ergebnis = LÄNGE(Markierung)»
«BESTIMMEN Ergebnis = LÄNGE(Feld)»
«BESTIMMEN Ergebnis = LÄNGE(Telefonnummer)»
```

INT reduziert eine Zahl auf die nächstkleinste Ganzzahl.

Zu dieser Funktion muß eigentlich nicht viel gesagt werden, dennoch ein Beispiel:

```
«BESTIMMEN Ergebnis =
INT(LÄNGE(Markierung)/2)»
```

legt die abgrundete halbe Anzahl der markierten Zeichen fest. Bei 9 Zeichen lautet das Ergebnis also 4. Um die Werte auf- bzw. abzurunden, addieren Sie einfach 0,5:

```
«BESTIMMEN Ergebnis =
INT((LÄNGE(Markierung)/2)+0,5)»
```

TEIL (auch als *MID* bekannt) entnimmt einer Textfolge eine bestimmte Anzahl Zeichen. Diese Funktion können Sie z.B. dazu nutzen, um nur Dateien mit einer bestimmten Endung zu laden:

```
«BESTIMMEN Dateityp =
TEIL(Dateiname,LÄNGE(Dateiname)/2)-3,3)»
«AWENN Dateityp = "SIK"»
```

Wie Sie sehen, erfordert *TEIL* drei verschiedene Elemente. Dabei ist in diesem Fall *LÄNGE(Dateiname)/2)-3* der Anfangspunkt für die drei letzten Zeichen.
Verketten – obwohl keine benannte Funktion wie die obigen – erlaubt dennoch die Verkettung von zwei oder mehreren Textfolgen. Der Umfang der verketteten Zeichen darf nicht mehr als 255 Zeichen betragen:

```
«BESTIMMEN Zeichenfolge = Zeichenfolge1
Zeichenfolge2 Zeichenfolge3 Zeichenfolge4»
```

Die folgenden neuen Operatoren erlauben eine größere Flexibilität der Ausdrücke, die Sie schreiben können: *UND*, *ODER* und *NICHT*. Damit sind jetzt auch lästige AWENN-Schachtelungen unnötig, so können Sie z.B. folgendes eingeben:

```
«AWENN a<b UND b<(c+d)»
«AWENN a>b ODER c>d»
```

Eine weitere Möglichkeit erspart Ihnen viel Arbeit beim Vergeben von Variablennamen: die sogenannte Aufstellung (Array). Sie können damit eine Matrix erstellen, um in einem einzigen Arbeitsgang eine ganze Gruppe verwandter Variablen zu definieren.

```
«BESTIMMEN Index = Ganzzahl»
«SOLANGE Index <= Maximum»
    •
«BESTIMMEN Aufstellung«Index» = Wert»
«BESTIMMEN Index = Index + Intervall»
    •
«ESOLANGE»
```

Nach der Initialisierung der Varaiblen *Index* mit einer ganzen Zahl, wird die SOLANGE-Schleife bis zu einem vorgegebenen Maximalwert durchlaufen. In der Schleife werden die Variablen, z.B. Aufstellung1, Aufstellung2, Aufstellung3 usw., durch Hochzählen des Index mit einem individuellen Wert belegt.

Noch ein paar interessante Kleinigkeiten

Auch Makros lassen sich wiederholen. Drücken Sie die Taste [F4], um die Makro-Ausführung zu wiederholen. In Word 4 wurde dadurch nur der letzte vom Makro ausgeführte Befehl wiederholt; in Word 5 ist es das gesamte Makro.

Wenn Die Makro-Ausführung infolge eines Fehlers unterbrochen wird, versucht Word, Ihnen wenn immer möglich die Ursache, des Abbruchs mitzuteilen und den Befehl oder die Anweisung zu identifizieren, durch den bzw. die der Abbruch verursacht wurde. Dies erleichtert Ihnen die Fehlerbehebung.

Es ist möglich, Variablen in den Text der Eingabeaufforderungen und Meldungen der Makro-Anweisungen *BESTIMMEN*, *ABFRAGE*, *MELDUNG* und *PAUSE* einzufügen. Der aktuelle Wert der Variable wird in der Meldung oder Eingabeaufforderung angezeigt. Sie können auch einen Teil des Makros »ausschalten«, indem Sie ihn zwischen *KOMMENTAR-EKOMMENTAR*-Anweisungen setzen.

Wenn Sie die Tastenbezeichnung <CTRL UMSCHALTEN UNT> in einem Makro verwenden – oder natürlich auch, wenn Sie dies über die Tastatur eingeben –, stellen Sie sicher, daß Word in jedem Fall zum Hauptbefehlsmenü geht, auch wenn es sich gerade im *Muster*-Menü befinden sollte.

Schlußbemerkung

Als alter Word-Anwender (bin seit der Version 1 dabei) freue ich mich über jede neue Version und natürlich über all das, was wieder weiter verbessert wurde. Diesmal freue ich mich besonders darüber, daß ich mich noch nicht ernsthaft einem DTP-Programm zugewendet habe. Diese mit Word 5 produzierte Ausgabe des *System Journals* gibt mir da recht, oder?!
Hartmut Niemeier

Der programmierbare Interruptcontroller 8259A:

Steuerung von Ein-/Ausgabegeräten

Im Gegensatz zu Softwareinterrupts, die von einem Programm ausgelöst werden, werden Hardwareinterrupts von einem elektrischen Signal, das von einem Ein-/Ausgabegerät, wie einer seriellen Schnittstelle oder einem Festplattencontroller kommt, initiiert. Der Mikroprozessor kann Hardwareinterrupts auch selbst auslösen. Die externen oder internen Hardwareinterrupts werden bei der Intel-Prozessorarchitektur nach Prioritäten bedient.

Die 8086-Mikroprozessoren-Familie (wo zu auch 8088, 8086, 80186, 80286 und 80386 gehören) reserviert die ersten 1024 Bytes des Arbeitsspeichers (die Adressen 0000:0000 bis 0000:03FFH) für eine Tabelle von 256 Interruptvektoren. Jeder dieser Vektoren ist ein 4 Byte langer Far-Zeiger auf eine zugehörige Interrupt-Serviceroutine (ISR), die bei der Bearbeitung des zugehörigen Interrupts ausgeführt wird. Das Design der 8086-Familie bedingt, daß einige Vektoren für spezielle Zwecke verwendet werden (*Tabelle 1*). Obwohl Intel die ersten 32 Interrupts reserviert hat, hat IBM die Interrupts 05H bis 1FH im Original-PC für eigene Zwecke verwendet. Die meisten, aber nicht alle dieser reservierten Vektoren, werden von der Software benötigt, nicht von der Hardware. Die von IBM anders definierten Interrupts sind in *Tabelle 2* aufgelistet.

▸▸ *Tabelle 1: Die von Intel reservierten Ausnahme-Interrupts*

Interrupt	Definition
00H	Division durch 0
01H	Einzelschritt
02H	Nicht maskierbarer Interrupt (NMI)
03H	Breakpunkt
04H	Überlauf
05H	Array-Grenzen-Überschreitung [1]
06H	Ungültiger Befehl [1]
07H	Coprozessor nicht verfügbar [2]
08H	Doppelfehler-Ausnahme [2]
09H	Coprozessor Segment-Überlauf [2]
0AH	Ungültiges Task-Zustandsregister [2]
0BH	Segment nicht vorhanden [2]
0CH	Stack-Ausnahme [2]
0DH	Generelle Schutzverletzung [2]
0EH	Seitenfehler [3]
0FH	Reserviert
10H	Coprozessor-Fehler [2]

[1] Nur für 80186, 80286 und 80386 Mikroprozessoren
[2] Nur für 80286 und 80386 Mikroprozessoren
[3] Nur für 80386 Mikroprozessoren

In der Mitte von *Tabelle 2* sehen Sie die acht Hardwareinterrupt-Vektoren (08H bis 0FH), die IBM im Original-PC verwendet hat. Diese acht Vektoren sind die maskierbaren Interrupts für die IBM-PC-Familie und Kompatible. Die zusätzlichen IRQ-Signale, über die der IBM PC/AT verfügt, werden später erläutert.

Die Interrupt-Konflikte, die in den *Tabellen 1* und *2* zu sehen sind, verursachten Kompatibilitätsprobleme bei der Entwicklung der 8086-Familie. Für eine vollständige Kompatibilität zu IBM muß der Gebrauch der Interruptvektoren analog zu IBM implementiert sein, selbst wenn es Konflikte mit dem Design der CPU gibt. Zum Beispiel wird beim Überschreiten der oberen oder unteren Grenze eines Arrays ein BOUND-Fehler ausgelöst. Dieser verursacht einen Interrupt 5H.

Aber der 80286-Prozessor, wie er in AT-kompatiblen Computern verwendet wird, schickt beim Auftreten eines BOUND-Fehlers den Bildschirminhalt an den Drucker, da IBM den Interrupt 5 für die Printscreen-Funktion verwendet.

Interrupt	Definition
05H	Print Screen
06H	Unbenutzt
07H	Unbenutzt
08H	Hardware IRQ0 (Timer) [1]
09H	Hardware IRQ1 (Tastatur)
0AH	Hardware IRQ2 (Reserviert) [2]
0BH	Hardware IRQ3 (COM2)
0CH	Hardware IRQ4 (COM1)
0DH	Hardware IRQ5 (Festplatte)
0EH	Hardware IRQ6 (Diskette)
0FH	Hardware IRQ7 (Drucker)
10H	Bildschirm-Funktionen
11H	Ausstattungsinformationen
12H	Speichergröße
13H	Diskettenein-/ausgabe-Funktionen
14H	Serielle Schnittstellen-Funktionen
15H	Kassetten/Netzwerk-Funktionen
16H	Tastatur-Funktionen
17H	Drucker-Funktionen
18H	ROM BASIC
19H	Erneutes Starten des Systemes
1AH	Datum/Uhrzeit setzen und lesen
1BH	Control Break
1CH	Timertick
1DH	Video Parameter-Zeiger
1EH	Disketten Parameter-Zeiger
1FH	Grafische Zeichentabelle

[1] IRQ = Interrupt Anforderungs Leitung
[2] Sehen Sie auch die Bilder 7 und 8

◄◄ *Tabelle 2: Die Benutzung der Interrupts durch IBM*

Arten von Hardware-Interrupts

Die 8086-Mikroprozessor-Familie kann drei Arten von Hardware-Interrupts bearbeiten. Zuerst gibt es da die internen, vom Mikroprozessor ausgelösten Interrupts (*Tabelle 1*). Zum zweiten existiert der nicht maskierbare Interrupt (Interrupt 2 oder NMI), der vom NMI-Signal erzeugt wird. Dieser Interrupt wird erzeugt, wenn beim 8088 und 8086 am Pin 17, beim 80286 am Pin 59 und beim 80386 am Pin 88 der Pegel auf high übergeht. Bei der IBM-PC-Familie (ausgenommen PCjr und Convertible) wird der NMI für Speicher-Paritätsfehler benötigt. Zum dritten gibt es die maskierbaren Interrupts, die normalerweise von Peripheriegeräten erzeugt werden.

Die maskierbaren Interrupts werden über einen programmierbaren Interruptcontroller (PIC) 8259A an den Prozessor geleitet. Wenn der PIC eine Interruptanforderung erhält, signalisiert er durch das Aktivieren der Interrupt-Request-Leitung (INTR) dem Mikroprozessor, daß ein Interrupt bedient werden muß. Dieser Artikel beschränkt sich auf die maskierbaren Interrupts und den 8259A, da Peripheriegeräte (Festplatten, serielle Schnittstellen und so weiter) so Zugriff auf das Interrupt-System erlangen.

Interrupt-Prioritäten

Die Intel-Mikroprozessoren haben ein eingebautes Prioritätssystem für die Behandlung gleichzeitig auftretender Interrupts. Höchste Priorität besitzen die internen Befehlsausführungs-Interrupts, wie Division durch 0 oder ungültiger Befehl, da die Priorität von der Interruptnummer bestimmt wird. Der Interrupt 0 hat die höchste Priorität, wogegen der Interrupt FFh nie bearbeitet wird, solange ein anderer Interrupt aktiv ist. Wenn jedoch die Interrupt-Bearbeitung aktiv ist (das Interrupt-Flag im Mikroprozessor gesetzt ist), besitzten alle Hardwareinterrupts höhere Priorität als die Softwareinterrupts (INT-Befehle). Die Prioritäts-Reihenfolge nach Interrupt-Nummern darf nicht mit der Priorität der externen Hardware-Interrupts verwechselt werden. Die hier besprochene zahlenmäßige Priorität gilt nur für die von der 8086-Mikroprozessor-Familie ausgelösten Interrupts und ist vollkommen unabhängig von den System-Interruptprioritäten der externen Komponenten.

Die Interrupt-Serviceroutinen

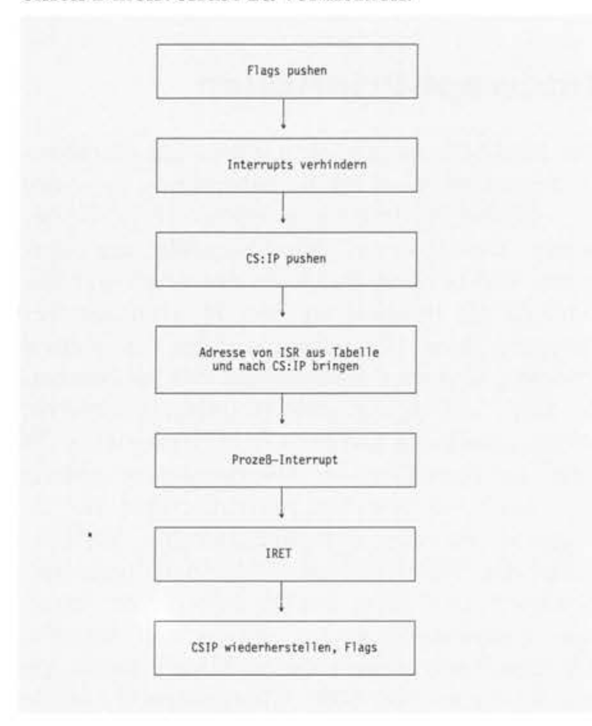
In der Regel brauchen Programmierer keine hardwareabhängigen Routinen schreiben, die die Hardware-Interrupts bedienen. Die Routinen des IBM-PC-BIOS zusammen mit denjenigen von MS-DOS genügen in der Regel. In einigen Fällen bieten jedoch MS-DOS und das BIOS nicht genügend Unterstützung für einen hohen Programmdurchsatz. Hierzu zählt vor allem die Kommunikationssoftware. Die Programmierer benötigen dann direkten Zugriff auf den 8259A und den universellen asynchronen Receiver und Transmitter (UART).

Es gibt zwei große Unterschiede zwischen den maskierbaren Interrupts und allen anderen Systemereignissen. Die maskierbaren Interrupts sind nicht vorhersagbar, und sie sind vergänglich. Normalerweise wird ein Hardware-Interrupt erzeugt, wenn ein Peripheriegerät das gesamte System benötigt. Falls das System nicht schnell genug antwortet, sind die Daten unwiederbringbar verloren.

Alle Dinge sind aber relativ, und dies gilt besonders für die Geschwindigkeit, mit der Interrupts bedient werden müssen. Nehmen wir zum Beispiel an, daß zwei Interrupts gleichzeitig auf-

treten. Einer stammt von einer seriellen Schnittstelle, die Daten mit 300 Baud empfängt, der andere ist von einer seriellen Schnittstelle, die Daten mit 9600 Baud empfängt. Die Daten vom ersten Kanal ändern sich 30 Millisekunden nicht. Die Daten des zweiten Kanals müssen aber innerhalb einer Millisekunde bearbeitet werden, um einen Datenverlust zu vermeiden.

▸ *Bild 1: Generelle Interruptausführung*



Unvorhersagbarkeit

Da die maskierbaren Interrupts als Folge eines externen Ereignisses wie dem Empfang eines Bytes über eine Kommunikationsschnittstelle auftreten, kann der Zeitpunkt nicht vorhergesagt werden. Sogar der Timer-Interrupt, der fest 18,2 mal je Sekunde auftritt, kann von einem Programm, das gerade läuft, nicht vorhergesagt werden.

Wegen dieser Gegebenheiten muß das System, falls es Interrupts zuläßt, für deren Behandlung ausgelegt sein. Daher müssen die Interrupts, falls sie nicht bedient werden können, gesperrt werden. Die 8086-Mikroprozessor-Familie bietet einen Befehl Set Interrupt Flag (STI), um Interrupts zuzulassen, und einen Befehl Clear Interrupt Flag (CLI), um die Interrupts zu sperren. Das Interrupt-Flag wird automatisch gelöscht, sobald ein Hardware-Interrupt bearbeitet wird. Der Interrupt-Handler sollte sobald wie möglich einen STI-Befehl enthalten, um höher priorisierte Interrupts zuzulassen.

Datenverlust

Wie wir schon gesehen haben, muß ein maskierbarer Interrupt sofort bedient werden, um einen Datenverlust zu vermeiden. Sofort ist aber relativ zu der Datentransferrate des Peripheriegerätes. Die Regel ist, daß die gerade anfallende Datenmenge bearbeitet werden muß (zumindest in einem Puffer gespeichert) bevor die nächsten Daten ankommen. Ausgenommen Festplatten, die immer sofortige Bearbeitung verlangen, sind Interrupts von Einheiten, die Daten empfangen kritischer, als Interrupts von Einheiten, die Daten senden.

Die Probleme des Datenverlusts bei Hardware-Interrupts werden durch die Einführung von Prioritäten für die Interrupts, die außerhalb des Mikroprozessors ausgelöst werden, vermieden. Geräten mit einer niedrigeren Datentransferrate wird eine niedrige Priorität zugewiesen. Die zeitkritischen Einheiten erhalten die höchsten Prioritäten.

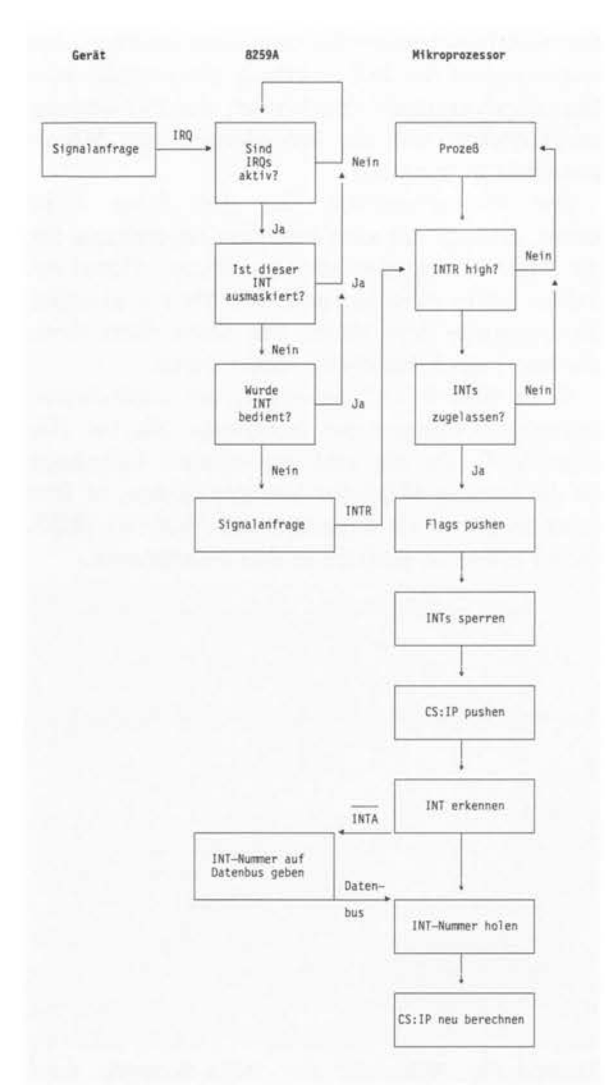
Maskierbare Interrupts

Der Mikroprozessor behandelt alle Interrupts (maskierbare, nicht maskierbare und Software) indem er die Flags auf dem Stack ablegt, das Interrupt-Flag löscht und die Inhalte der Register CS und IP auf dem Stack ablegt.

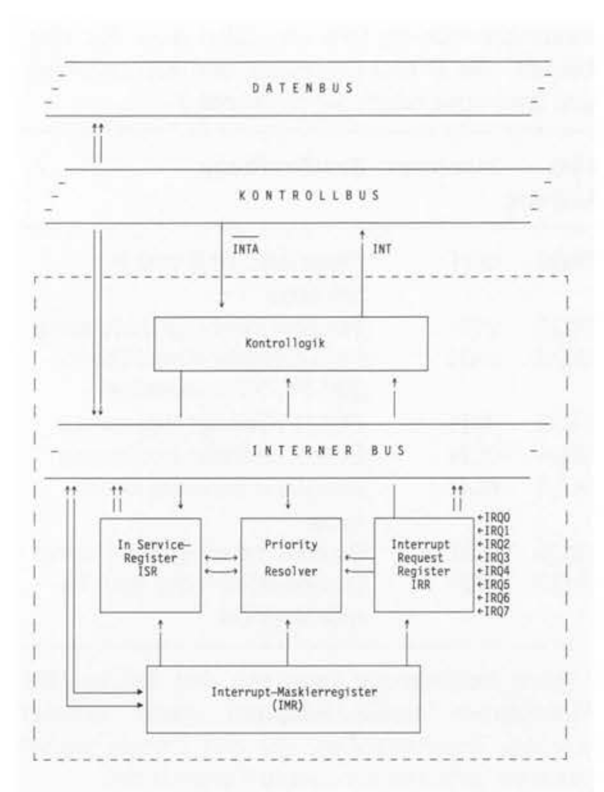
Der Mikroprozessor holt sich dann die Interrupt-Nummer vom Datenbus, multipliziert sie mit 4 (die Größe eines Vektors in Bytes), und verwendet das Ergebnis als Offset in der Interrupt-Vektor-Tabelle, die sich in dem untersten (Segment 0000), 1 Kbyte großen Speicherbereich befindet. Die 4-Byte-Adresse an dieser Stelle wird dann als neuer Wert in die Register CS und IP geladen (*Bild 1*).

Externen Geräten werden feste Interrupt-Leitungen (IRQ's) zugeteilt, die mit dem 8259A verbunden sind. Dies wird weiter unten noch erläutert. Benötigt ein Gerät eine Bearbeitung, so sendet es über die IRQ-Leitung ein Signal an den PIC. Der PIC, der wie ein Erfüllungsgehilfe für das Gerät arbeitet, funktioniert wie in *Bild 2* gezeigt. Er bewertet die Serviceanfrage, und falls sie richtig ist, aktiviert er die INTR-Leitung des Prozessors. Der Mikroprozessor prüft dann, ob Interrupts zulässig sind, ob also das Interrupt-Flag gesetzt ist. Ist dies der Fall, werden die Flags auf dem Stack abgelegt, das Interrupt-Flag gelöscht, und die Inhalte der Register CS und IP auf dem Stack abgelegt.

Der Mikroprozessor beantwortet die Interrupt-Anforderung durch das Aktivieren der Interrupt-Acknowledge-Leitung (INTA). Der 8259A gibt dann die Interrupt-Nummer auf dem Datenbus aus. Der Mikroprozessor holt sich die Interrupt-Nummer vom Datenbus und bedient den Interrupt. Vor dem Ausführen des IRET-Befehls muß die Interrupt-Serviceroutine dem 8259A eine »End of Interrupt«-(EOI)-Mitteilung senden. Dies wird durch Ausgeben des Wertes 20h an die Adresse 20h erledigt (die Übereinstimmung der Nummern ist rein zufällig).



◂◂ *Bild 2: Maskierbare Interrupt-Routine*



◂ *Bild 3: Blockschaltbild des programmierbaren Interrupt Controllers 8259A*

Der 8259A (*Bild 2*) besitzt viele interne Bestandteile, von denen einige durch die Software beeinflußbar sind. Nur die Einstellungen der IBM-PC-Familie werden hier behandelt.

Drei Register beeinflussen das Bearbeiten der maskierbaren Interrupts: das Interrupt Request Register (IRR), das In Service Register (ISR) und das Interrupt Mask Register (IMR).

Das IRR-Register wird benötigt, um festzustellen, für welches Gerät gerade eine Interruptanforderung vorliegt. Aktiviert ein peripheres Gerät seine IRQ-Leitung, um dem 8259A mitzuteilen, daß es Bedienung benötigt, so wird im IRR ein Bit gesetzt, das der Interrupt-Ebene des Geräts entspricht.

Das ISR gibt die Interrupt-Ebene an, die gerade bedient wird. Das ISR-Bit wird gesetzt, wenn die CPU (durch INTA) den Interrupt zuläßt, und sich die Interrupt-Nummer auf dem Datenbus befindet. Das ISR-Bit bleibt bis zu der EOI-Sequenz gesetzt.

Das IMR-Register kann beschrieben und gelesen werden (an Adresse 21H). Hier können einzelne Interrupts gesperrt werden. Ist ein Bit in diesem Register gesetzt, ist die zugehörige IRQ-Leitung maskiert und wird nicht bedient, bis das Bit wieder gelöscht wird. So kann ein Interrupt gesperrt, die anderen aber weiterhin bearbeitet werden.

Der vierte große Block im *Bild 3*, bezeichnet als Priority Solver, ist ein komplexes Gebilde, welches das Herz des 8259A darstellt. Diese Komponente verknüpft die Inhalte der IMR, ISR und IRR, um zu entscheiden, falls eine Interruptanforderung vorliegt, ob diese bearbeitet werden soll, und dann die INTR-Leitung zum Prozessor aktiviert. Der Priority Solver kann sehr vielfältig programmiert werden. Hier wird aber nur der in den IBM PCs und in Kompatiblen verwendete Modus beschrieben.

Die IRQ-Ebenen

Stehen zwei oder mehrere unbediente Hardware-Interrupt-Anforderungen an, so entscheidet der 8259A, welche zuerst bedient wird. Der Standardmodus des PIC ist der vollständig geschachtelte Modus, bei dem die IRQ-Leitungen in einer festen Reihenfolge priorisiert sind. Nur IRQ-Leitungen mit einer höheren Priorität, als die gerade bearbeitete, dürfen Interrupts auslösen.

Die höchste Priorität besitzt die Leitung IRQ0, die niedrigste IRQ7. Wenn also gerade ein Interrupt 09h (ausgelöst durch IRQ1) bearbeitet wird, kann nur ein Interrupt 08h (ausgelöst durch IRQ0) ihn unterbrechen. Alle anderen Interrupts werden verzögert, bis die Serviceroutine des Interrupts 09h beendet ist und EOI ausgegeben hat.

Design mit 8 Ebenen

Die IBM PC und PC/XT (und kompatiblen) Computer haben acht IRQ-Leitungen zum PIC – IRQ0 bis IRQ7. Diese Interrupt-Leitungen lösen die

Interrupts 08h bis 0Fh aus (also 8 + der IRQ-Ebene). Die 8 IRQ-Leitungen und die zugehörigen Interrupts sehen Sie in *Tabelle 3.*

▸ *Tabelle 3: Interrupt-Zuordnung bei 8 Ebenen*

IRQ Leitung	Interrupt	Beschreibung
IRQ0	08H	Timertick, 18,2 mal je Sekunde
IRQ1	09H	Tastatur benötigt Bedienung
IRQ2	0AH	Ein-/Ausgabe-Kanal (beim IBM PC/XT unbenutzt)
IRQ3	0BH	COM1 benötigt Bedienung
IRQ4	0CH	COM2 benötigt Bedienung
IRQ5	0DH	Festplatte benötigt Bedienung
IRQ6	0EH	Diskette benötigt Bedienung
IRQ7	0FH	Datenanforderung vom Paralleldrucker [1]

[1] Diese Anforderung kann von den älteren IBM Monochrom-/Drucker-Adaptern nicht erzeugt werden. Druckertreiber, die auf diesem Signal basieren, arbeiten mit diesen Karten nicht.

Design mit 16 Ebenen

Im IBM PC/AT gibt es weitere acht IRQ-Ebenen. Dies wurde durch einen zweiten PIC, der kaskadiert ist, erreicht. So sind hier 16 Prioritätsebenen vorhanden.

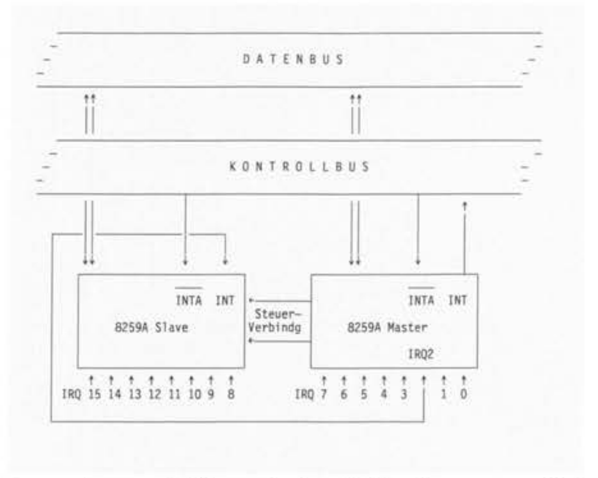
▸ *Tabelle 4: Interrupt Zuordnung bei 16 Ebenen*

IRQ Leitung	Interrupt	Beschreibung
IRQ0	08H	Timertick, 18,2 mal je Sekunde
IRQ1	09H	Tastatur benötigt Bedienung
IRQ2	0AH	INT vom zweiten 8259A
IRQ8	70H	Echtzeituhr
IRQ9	71H	Software-Umleitung auf IRQ2
IRQ10	72H	Reserviert
IRQ11	73H	Reserviert
IRQ12	74H	Reserviert
IRQ13	75H	Numerischer Coprozessor
IRQ14	76H	Festplattenkontroller
IRQ15	77H	Reserviert
IRQ3	0BH	COM2 benötigt Bedienung
IRQ4	0CH	COM1 benötigt Bedienung
IRQ5	0DH	Datenanforderung von LPT2
IRQ6	0EH	Diskette benötigt Bedienung
IRQ7	0FH	Datenanforderung von LPT1

Die Kaskadierung besteht darin, daß die INT-Leitung des Slave 8259A mit der IRQ2-Leitung des Master 8259A verbunden ist, anstatt mit dem Mikroprozessor. Löst ein Gerät, das mit einer IRQ-Leitung des Slave 8259A verbunden ist, einen Interrupt aus, so wird die INT-Leitung des Slave aktiviert, was wiederum eine Interrupt-Anforderung an der IRQ2-Leitung des Masters auslöst. Dies veranlaßt den Master, die INT-Leitung zu aktivieren, und die Anforderung dem Mikroprozessor mitzuteilen.

Der Mikroprozessor, der den Slave nicht kennt, erzeugt nur eine Interrupt-Bestätigung für die Anforderung des Masters. Dieses Signal erhalten beide 8259As, und der Master übergibt die Kontrolle dem Slave. Der Slave führt dann die Interrupt-Behandlung weiter durch.

Beim IBM PC/AT erzeugen die zusätzlichen Interrupt-Leitungen die Interrupts 70h bis 77h (*Tabelle 4*). Da die acht zusätzlichen Leitungen an die Leitung IRQ2 des Masters führen, ist ihre Priorität höher als diejenige von IRQ3 bis IRQ7. Dieser Effekt ist grafisch in *Bild 4* dargestellt.



▸▸ *Bild 4: Eine grafische Darstellung der IRQ Prioritäten durch die Kaskadierung*

Anmerkung: Während der INTA-Sequenz wird das entsprechende ISR-Bit in beiden 8259As gesetzt. Deshalb müssen zwei EOI gesendet werden, um den Interrupt zu beenden. Eines an den Master, und eines an den Slave.

Programmierung der Hardware-Interrupts

Jedes Programm, das einen Interrupt-Vektor ändert, muß diesen vor der Rückkehr zu DOS oder seinem aufrufenden Prozeß zurücksetzen. Jedes Programm, das einen Hardware-Interrupthandler ersetzt, muß alle Aufgaben des originalen Interrupthandlers erledigen. Es muß die Interrupts wieder freigeben, dem Interruptcontroller EOI senden und so weiter. Nichtbeachtung dieser Regeln führte bei vielen Programmierern schon zu mehrstündigen Fehlersuchen.

Wird ein Interrupthandler vollständig durch einen eigenen ersetzt, so muß der originale Interrupt-Vektor gespeichert werden, so daß er am Programmende wieder zurückgesetzt werden kann. Obwohl es möglich ist, den 4 Byte großen Vektor im RAM direkt umzusetzen (und viele Programme tun dies auch), sollte man dies in Anbetracht auf Multitasking-Erweiterungen, oder spätere Versionen von DOS nicht tun. Die einzige empfehlenswerte Technik für das Lesen und Set-

zen von Interrupt-Vektoren ist der Aufruf der entsprechenden MS-DOS-Funktionen. Die Funktionen 25h (Setzen des Interrupt-Vektors) und 35h (Lesen des Interrupt-Vektors) des Interrupts 21h erledigen dies.

Nachdem der originale Vektor gespeichert wurde, kann er durch einen Far-Zeiger, der auf die Ersatzroutine zeigt, ersetzt werden. Die neue Routine muß mit einem IRET-Befehl enden. Sie sollte auch alle Prozessorregister und Zustände am Beginn retten und vor dem Ende wiederherstellen.

Ersatz-Interrupthandler

Denken Sie an ein Programm, das viele mathematische Berechnungen mit zufälligen Werten durchführt. Um einen abnormalen Abbruch durch den MS-DOS-Interrupt 00h-Handler zu vermeiden, wenn ein DIV- oder IDIV-Befehl mit dem Divisor 0 bearbeitet wird, kann der Programmierer die Interrupt-Routine 00h (Division durch 0) durch eine Routine ersetzen, die den Anwender informiert, was geschehen ist, und dann die Bearbeitung ohne Abbruch fortsetzt. Das COM-Programm DIVZERO.ASM (*Listing 1*) erledigt dies.

Unterstützende Interrupthandler

In vielen Fällen unterstützt ein Interrupthandler einen existierenden, anstatt ihn zu ersetzen. Die zusätzliche Routine kann die Daten bearbeiten, bevor sie sie an die eigentliche Interrupt-Routine weitergibt. Sie kann aber auch die Bearbeitung nach dem Aufruf der originalen Interrupt-Routine erledigen. Diese Fälle verlangen ein unterschiedliches Kodieren der Handler.

Soll die zusätzliche Routine vor dem originalen Handler ausgeführt werden, so ruft die zusätzliche Routine den Original-Handler am Ende auf. Dieser Aufruf kann indirekt über den am Anfang gespeicherten Original-Interrupt-Vektor erfolgen. Ein zusätzlicher Interrupt-Handler, der zum Beispiel beim Auftreten eines Interrupts 08h einen internen Zähler erhöht, kann wie in *Listing 2* aussehen.

Der zusätzliche Handler muß alle Register und Maschinenzustände belassen, außer diejenigen, die er gezielt ändern will. Hier ist dies der Wert von myflag (und das Flag-Register, das aber durch den Interrupt-Vorgang gerettet wird). Die Register und die Maschinenzustände müssen vor dem Aufruf des Original-Handlers wieder hergestellt werden.

Die Sache ist etwas komplizierter, wenn die zusätzliche Interrupt-Routine nach der originalen Interrupt-Routine in Aktion treten soll, und besonders, wenn die zusätzliche Routine nicht wiedereintrittsfest ist. In diesem Fall muß der zusätzliche Handler geschachtelte Interrupts verhindern. So darf der zusätzliche Handler nicht durch einen Interrupt unterbrochen werden, selbst wenn der Original-Handler vorher ein EOI ausgegeben hat. Statt der vorhergehenden Interrupt-Routine 08h kann der Programmierer das Programmstück in *Listing 3* verwenden, um myflag als Semaphore zu benutzen und mit der Funktion XCHG zu testen.

Beachten Sie, daß solch ein Interrupt-Handler den Original-Interrupt-Aufruf simulieren muß, indem er zuerst die Flags auf dem Stack ablegt (PUSHF) und dann einen Far-CALL ausführt. Die auf dem Stack abgelegten Flags werden durch den IRET-Befehl der Original-Handler-Routine wieder geladen. Nach der Rückkehr vom Original-Handler kann die zusätzliche Routine den Maschinenzustand retten und seine eigene Bearbeitung durchführen. Das Ausführen des IRET-Befehls führt dann zum unterbrochenen Programm zurück.

Die Flags in der zusätzlichen Routine brauchen nicht gerettet zu werden, da sie automatisch vom IRET-Befehl mit dem Originalwert geladen werden. Wegen dieser Art der Interrupt-Behandlung sollte die Interrupt-Routine nicht von Informationen der Flags abhängen, und sie kann ebenso keine Informationen ins Flag-Register zurückliefern. Beachten Sie auch, daß der Original-Handler (der durch den indirekten CALL aufgerufen wird) den Interrupt abgeschlossen hat, da er das EOI an den 8259A gesendet hat. Deshalb ist der Zustand der Maschine nicht derselbe wie im ersten Beispiel.

Beim Zurückspeichern des geretteten Interrupt-Vektors braucht das Programm nur den neuen Vektor (in der Vektortabelle) durch die gespeicherte Kopie zu ersetzen. Ist die Ersatz-Interruptroutine Teil einer Applikation, so muß der Original-Interrupt-Vektor bei jeder Möglichkeit des Programmendes (Control-C, Control-Break und kritischer Programmabbruch im Fehlerfalle) zurückgespeichert werden. Unterbleibt dies, so ist der Systemabsturz gewiß. Obwohl dies nicht sofort der Fall sein muß, trifft das doch ein, sobald ein anderes Programm den Speicherbereich des Interrupt-Handler-Codes belegt.

Zusammenfassung

Handler-Routinen für Hardware-Interrupts, obwohl sie kein Bestandteil von DOS sind, werden doch von vielen MS-DOS-Applikationen verwendet. Routinen dieses Typs spielen in der Funktionalität des IBM Personalcomputers eine große Rolle und können den Durchsatz eines Programms, wenn sie gut programmiert sind, erheblich verbessern. Unter gewissen Gegebenheiten gibt es keine andere praktikable Methode.

Jim Kyle / Chip Rabinowitz

*Listing 1:
Der Ersatz-Interrupt-Handler für die Division durch 0*

```
        name    divzero
        title   'DIVZERO - Interrupt 00H Handler'
;
;DIVZERO.ASM: Demonstration Interrupt 00H Handler
;This code is specific to 80286 and 80386 microprocessors.
;To assemble, link, and convert to a COM file:
;
;       MASM DIVZERO
;       LINK DIVZERO
;       EXE2BIN DIVZERO.EXE DIVZERO.COM
;       DEL DIVZERO.EXE
;

cr      equu    0dh     ; ASCII carriage return
lf      equ     0ah     ; ASCII linefeed
eos     equ     '$'     ; end of string marker

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:_TEXT,es:_TEXT,ss:_TEXT

        org     100h

entry:  jmp     start   ; skip over data area

intmsg  db      'Divide by Zero Occurred!',cr,lf,eos

divmsg  db      'Dividing '     ; message used by demo
par1    db      '0000h' ; dividend goes here
        db      ' by '
par2    db      '00h'   ; divisor goes here
        db      ' equals '
par3    db      '00h'   ; quotient here
        db      'remainder'
par4    db      '00h'   ; and remainder here
        db      cr,lf,eos

oldint0 dd      ?       ; save old Int00H vector

intflag db      0       ; nonzero if divide by
                        ; zero interrupt occurred

oldip   dw      0       ; save old IP value

;
;The routine 'int0' is the actual divide by zero interrupt handler.
;It gains control whenever a divide by zero or overflow occurs. Its
;action is to set a flag and then increment the instruction pointer
;saved on the stack so that the failing divide will not be reexecuted
;after the IRET.
;
;In this particular case we can call MS-DOS to display a message during
;interrupt handling because the application triggers the interrupt
;intentionally. Thus, it is known that MS-DOS or other interrupt
;handlers are not in control at the point of interrupt.
;

int0:   pop     cs:oldip                ;capture instruction pointer

        push    ax
        push    bx
        push    cx
        push    dx
        push    di
        push    si
        push    ds
        push    es

        push    cs      ; set DS = CS
        pop     ds

        mov     ah,09h  ; print error message
        mov     dx,offset _TEXT:intmsg
        int     21h

        add     oldip,2 ; bypass instruction causing
                        ; divide by zero error

        mov     intflag,1       ; set divide by 0 flag

        pop     es      ; restore all registers
        pop     ds
        pop     si
        pop     di
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        push    cs:oldip        ; restore instruction pointer

        iret            ; return from interrupt


;
;The code beginning at 'start' is the application program. It alters
;the vector for Interrupt 00H to point to the new handler, carries
;out some divide operations (including one that will trigger an
;interrupt) for demonstration purposes, restores the original
;contents of the Interrupt 00H vector, and then terminates.
;

start:  mov     ax,3500h        ; get current contents
        int     21h     ; ofInt 00H vector

                        ; save segment:offset
                        ; of previous Int 00H handler
        mov     word ptr oldint0,bx
        mov     word ptr oldint0+2,es

                        ; install new handler ...
        mov     dx,offset int0  ; DS:DX = handler address
        mov     ax,2500h        ; call MS-DOS to set
        int     21h     ; Int 00H vector

                        ; now our handler is active,
                        ; carry out some test divides.

        mov     ax,20h  ; test divide
        mov     bx,1    ; divide by 1
        call    divide

        mov     ax,1234h        ; test divide
        mov     bx,5eh  ; divide by 5EH
        call    divide

        mov     ax,5678h        ; test divide
        mov     bx,7fh  ; divide by 127
        call    divide

        mov     ax,20h  ; test divide
        mov     bx,0    ; divide by 0
        call    divide  ; (triggers interrupt)

                        ; demonstration complete,
                        ; restore old handler

        lds     dx,oldint0      ; DS:DX = handler address
        mov     ax,2500h        ; call MS-DOS to set
        int     21h     ; Int 00H vector

        mov     ax,4c00h        ; final exit to MS-DOS
        int     21h     ; with return code = 0

;
;The routine 'divide' carries out a trial division, displaying the
;arguments and the results. It is called with AX = dividend and
;BL = divisor.
;

divide  proc    near

        push    ax      ; save arguments
        push    bx

        mov     di,offset par1  ; convert dividend to
        call    wtoa    ; ASCII for display

        mov     ax,bx   ; convert divisor to
        mov     di,offset par2  ; ASCII for display
        call    btoa

        pop     bx      ; restore arguments
        pop     ax

        div     bl      ; perform the division
        cmp     intflag,0       ; divide by zero detected?
        jne     nodiv   ; yes, skip display

        push    ax      ; no, convert quotient to
        mov     di,offset par3  ; ASCII for display
        call    btoa

        pop     ax      ; convert remainder to
        xchg    ah,al   ; ASCII for display
        mov     di,offset par4
        call    btoa

        mov     ah,09h  ; show arguments, results
        mov     dx,offset divmsg
        int     21h

nodiv:  mov     intflag,0       ; clear divide by 0 flag
        ret             ; and return to caller

divide  endp


wtoa    proc    near    ; convert word to hex ASCII
                        ; call with AX = binary value
                        ;           DI = addr for string
                        ; returns AX, CX, DI destroyed

        push    ax      ; save original value
        mov     al,ah
        call    btoa    ; convert upper byte
        add     di,2    ; increment output address
        pop     ax
        call    btoa    ; convert lower byte
        ret             ; return to caller

wtoa    endp
```

```
btoa    proc        near        ; convert byte to hex ASCII
                            ; call with AL = binary value
                            ;           DI = addr to store string
                            ; returns AX, CX destroyed

        mov     ah,al   ; save lower nibble
        mov     cx,4    ; shift right 4 positions
        shr     al,cl   ; to get upper nibble
        call    ascii   ; convert 4 bits to ASCII
        mov     [di],al ; store in output string
        mov     al,ah   ; get back lower nibble

        and     al,0fh  ; blank out upper one
        call    ascii   ; convert 4 bits to ASCII
        mov     [di+1],al       ; store in output string
        ret             ; back to caller

btoa    endp


ascii   proc    near    ; convert AL bits 0-3 to
                        ; ASCII (0...9,A...F)
        add     al,'0'  ; and return digit in AL
        cmp     al,'9'
        jle     ascii2
        add     al,'A'-'9'-1    ; "fudge factor" for A-F
ascii2  ret             ; return to caller

ascii   endp

_TEXT   ends

        end     entry
```

◄◄ Listing 1: Ende

```
        .
        .
        .

myflag  dw      ?       ; variable to be incremented
                        ; on each timer-tick interrupt

oldint8 dd      ?       ; contains address of previous
                        ; timer-tick interrupt handler

        .
        .
        .               ; get the previous contents
        .               ; of the Interrupt 08H vector...
        mov     ax,3508h        ; AH = 35H (Get Interrupt Vector)
        int     21h     ; AL = Interrupt number (08H)
        mov     word ptr oldint8,bx     ; save the address of the
        mov     word ptr oldint8+2,es   ; previous Int 08H Handler

        mov     dx,seg myint8   ; put address of the new
        mov     ds,dx   ; interrupt handler into DS:DX
        mov     dx,offset myint8        ; and call MS-DOS to set vector
        mov     ax,2508h        ; AH = 25H (Set Interrupt; Vector)
        int     21h     ; AL = Interrupt number (08H)
        .
        .
        .

myint8:         ; this is the new handler
                ; for Interrupt 08H

        inc     cs:myflag       ; increment variable on each
                        ; timer-tick interrupt

        jmp     dword ptr cs:[oldint8]  ; then chain to the
                        ; previous interrupt handler
```

◄◄ Listing 2: Beispiel für einen zusätzlichen Interrupt-Handler

```
myint8:                 ; this is the new handler
                        ; for Interrupt 08H

        mov     ax,1    ; test and set interrupt-
        xchg    cs:myflag,ax    ; handling-in-progress
                        ; semaphore

        push    ax      ; save the semaphore

        pushf           ; simulate interrupt,
                        ; allowing the previous
        call    dword ptrcs:oldint8     ; handler for the
                        ; Interrupt 08H
                        ; vector to run

        pop     ax      ; get the semaphore back
        or      ax,ax   ; is our interrupt handler
                        ; already running?

        jnz     myint8x ; yes, skip this one

        .               ; now perform our interrupt
        .               ; processing here...
        .

        mov     cs:myflag,0     ; clear the interrupt-
                        ; handling-in-progress
                        ; flag

myint8x:
        iret            ; return from interrupt
```

◄ Listing 3: Ersatz-Interrupt-Handler mit Semaphoren

Siemens HighScan 800 im Kurztest:

Ein Scanner für alle Zwecke

Ein für viele Entwickler leidiges Thema sind die Dokumentationen zu den fertiggestellten Produkten. Vielfach ist nicht nur das »Wie« eine Frage – ein Entwickler kennt sein Programm ja in- und auswendig, wozu also diese detaillierten Beschreibungen –, sondern auch das »Womit«. *Eine* Antwort darauf wird in diesem Heft im Artikel über Word 5 gegeben. Eine sinnvolle Ergänzung dazu stellt ein Scanner dar, mit dem bereits angefertigte Zeichnungen oder auch maschinengeschriebene Texte (das soll's noch geben) in den Computer gelangen.

Die Scannerfamilie von Siemens besteht aus den Modellen HighScan 400, 600 und 800, mit eben diesen Auflösungen, gemessen in dpi (dots per inch). Ihre hohe Auflösung – vor allem der Modelle 600 und 800 – bringt sowohl bei graphischen Anwendungen als auch bei der Schrifterkennung Vorteile.

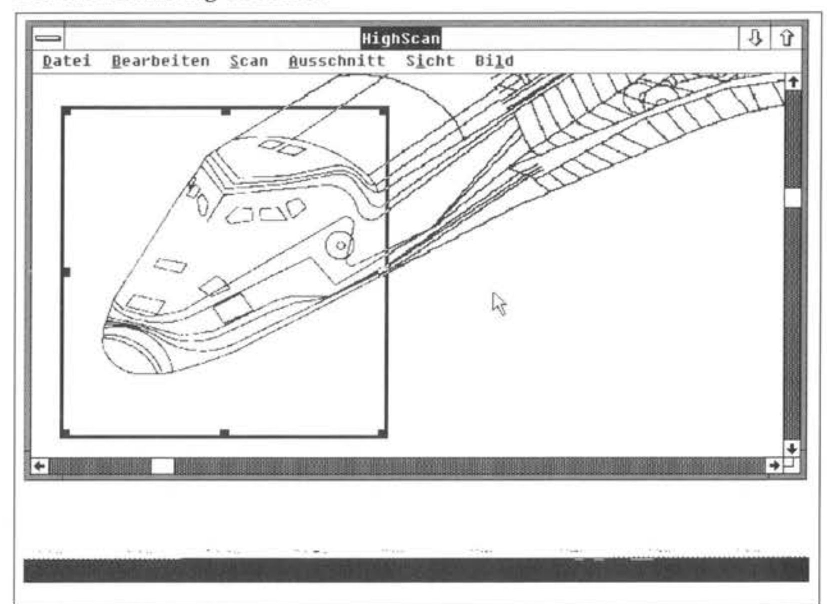


Bild 1: Aus der mit Prescan als reines Schwarz/Weiß-Bild erfaßten Vorlage wird ein Ausschnitt gewählt.

Mit der Auflösung des HighScan 800, den wir ausprobieren konnten, will Siemens in bisher nicht erreichte Regionen vorstoßen. Doch das sei gleich an dieser Stelle gesagt: Wenn Sie diese Auflösung nutzen wollen, sollte Ihre Festplatte nicht nur eine Kapazität von mindestens 100 Mbyte haben, sie sollte auch noch größtenteils frei sein. Bei einer Schwarz/Weiß-Darstellung der größtmöglichen Fläche werden insgesamt 65.280.000 Punkte erfaßt, das sind rund 8 Mbyte. Wird gleiches in den möglichen 64 Graustufen erfaßt, erhöht sich der Speicherbedarf auf 48 Mbyte – wohlgemerkt, für *ein* (allerdings sehr großes) Bild.

Ein weiterer Einsatzschwerpunkt der hochauflösenden HighScan-Modelle ist die optische Zeichenerkennung (OCR). So lassen sich schon mit einer physikalischen Auflösung von 400 dpi etwa auch Petitschriften im 6-Punkt-Bereich verarbeiten, z.B. bei der lernfähigen OCR-Software Optopus. Auch bei einer vortrainierten OCR-Software, wie z.B. Recognita, zeigt sich, wie sehr es beim Einlesen durch den Scanner auf eine möglichst hohe Auflösung ankommt. Auf dieses Produkt, das uns in einer Demo-Version vorlag, werden wir später noch eingehen.

Die nüchterne Hardware

Der Scanner präsentiert sich in einem nüchternen, kantigen, grauen, aber funktionellen Gehäuse. Die Vorlagen werden unter den nach oben schwenkbaren Deckel auf eine Glasplatte gelegt. Die Verbindung mit dem Rechner stellen eine mitgelieferte SCSI-Schnittstelle auf einer kurzen

Karte sowie ein dickes Kabel her. Also, hier muß ein wenig geschraubt und gesteckt werden, was aber auch für einen Hardware-Laien kein Problem sein dürfte.

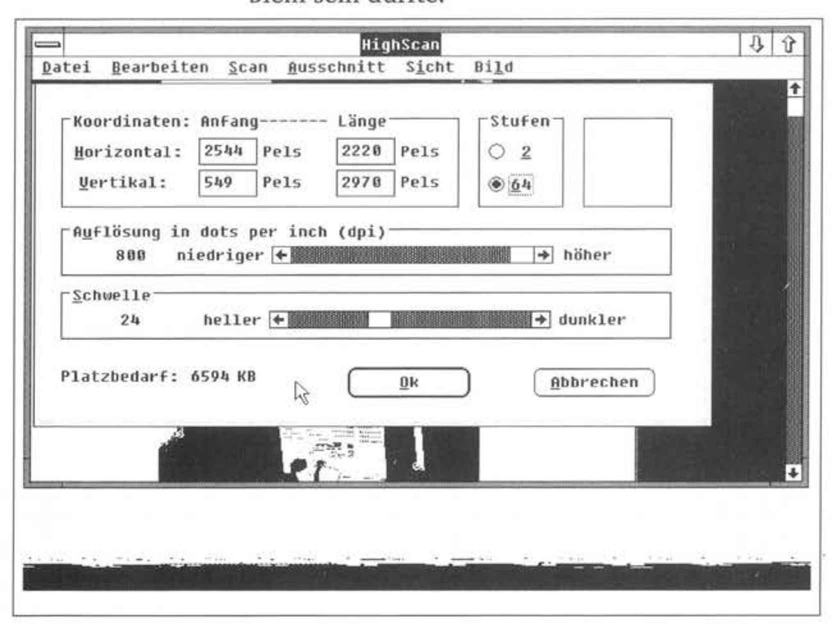


Bild 2:
Hier stellen Sie die Parameter ein, mit denen die Vorlage gescannt werden soll. Dabei wird der Platzbedarf auf der Festplatte gleich mit angezeigt.

Der Scanner kann Vorlagen bis zu einer Größe von 8,5 × 12 Zoll erfassen. Bei der Abtastung wird die Vorlage mit LEDs beleuchtet und die reflektierte Helligkeit von einem CCD-Sensor (Charge Coupled Device) »gemessen« und in elektrische Signale umgesetzt. Diese werden in Bits und Bytes umgewandelt und in dem 2 Mbyte großen Bildspeicher des Scanners zwischengelagert, bevor sie über die SCSI-Schnittstelle in den Rechner geschaufelt werden.

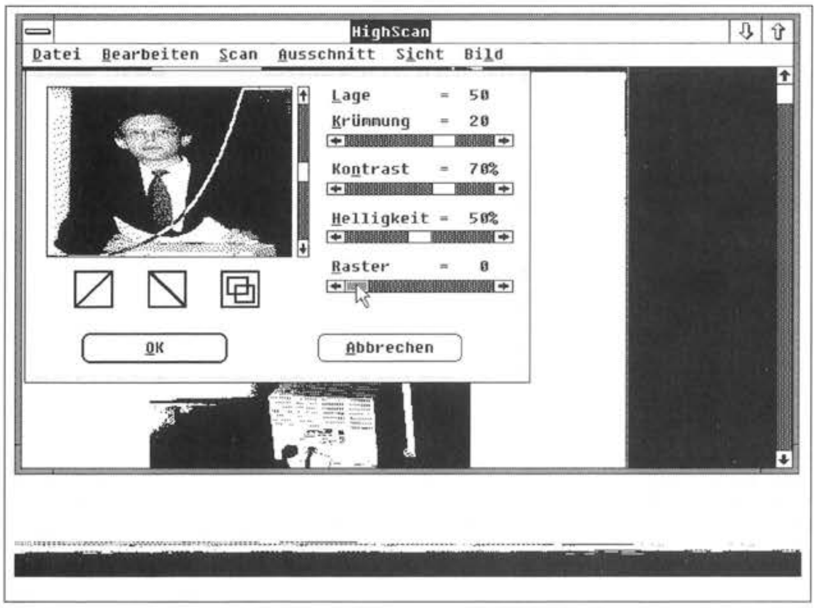


Bild 3:
Mit dem Menü Bild-Einstellen können Sie verschiedene Effekte erreichen.

Die vielfältige Software

Für die High-Scanner von Siemens werden eine ganze Reihe Softwareprodukten am Markt angeboten. Außerdem können die abgetasteten und aufbereiteten Text-, Bild- und Grafikdaten in vielen Standardprogrammen, wie z.B. Word oder PageMaker, weiterverarbeitet werden.

Pünktchen für Pünktchen

Der eigentliche Vorgang des Scannens ist denkbar einfach, nicht zuletzt aufgrund der unter Windows laufenden HighScan-Software.

Um den Bildausschnitt zu bestimmen – meist braucht man nur einen Teil der Vorlage – wird erst einmal vorgescannt. Jetzt kann man den Bildausschnitt vergrößern, stufenweise bis zu Faktor 16 (*Bild 1*).

Nun kann man sich entscheiden, ob man eine Strichzeichnung in zwei Stufen oder ein Foto in 64 Stufen (Halbton), wie in *Bild 2* dargestellt, scannen will. Die Auflösung kann in Stufen von 1 dpi von 50 bis 800 dpi eingestellte werden. Die Zeichnung vom Space Shuttle haben wir natürlich nur in zwei Stufen in den Auflösungen 200, 400, 600 und 800 dpi abgetastet, wobei die Datei mit der letzteren Auflösung einen Umfang von knapp 180 Kbyte hat.

Die damit erzielten Ergebnisse sind in *Bild 4* bis *6* bis 600 dpi dargestellt. Wir haben darauf verzichtet, auch noch die höchste Auflösung zu zeigen, da der Laserbelichter sonst unnötig lange wegen dieses einen Bilds beschäftigt gewesen wäre. Ohnehin mußte der Druckertreiber von Word umgeschrieben werden, da er die Auflösung auf 300 dpi begrenzt.

Eingescannte Halbtonvorlagen können Sie in einer »elektronischen Dunkelkammer« nachbearbeiten (*Bild 3*). Hier läßt sich die Bildqualität mit den verschiedensten Mitteln wie Krümmung, Kontrast, und Helligkeit beeinflussen.

Die so bearbeiteten Bilder bzw. Ausschnitte (bis zu 99 sind möglich) können Sie in verschiedenen gängigen Formaten abspeichern: TIFF (Word, Pagemaker), IMG (Ventura Publisher), PCX (Paintbrush), MSP (Windows Paint). Beim Abspeichern im TIFF-Format fiel auf, daß die High-Scan-Software die Datenkompression noch nicht unterstützt und auch die Bildgröße nicht mit festhält.

Ich erkenne Ihren Charakter

Die optische Zeichenerkennenung (OCR, optical character recognition) – ein Einsatzbereich, der von etwa 30% der Scannerbesitzer genutzt wird – haben wir mit dem ungarischen windowsähnlichen Programm Recognita (Preis etwa DM 3000,-) mit ausprobiert.

Schlußbemerkung

Der High-Scan 800 (Preis ca. DM 14.000) war leicht zu bedienen und lieferte gute Ergebnisse sowohl bei der Bildverarbeitung als auch bei der Zeichenerkennung. Bei der Anschaffung eines solchen Geräts sollten Sie jedoch bedenken, ob Sie die hohe Auflösung nutzen können, oder vielleicht mit einem Modell 600 oder 400 auskommen.

Hartmut Niemeier

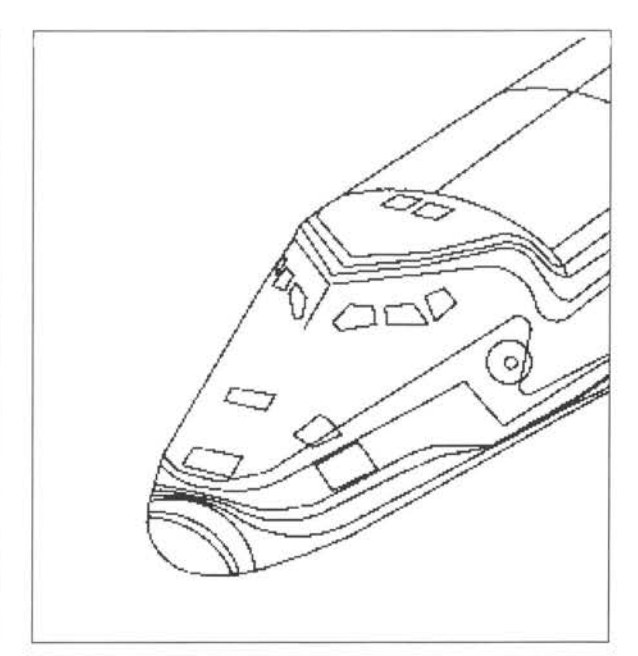


Bild 4: Gescannt mit 200 dpi.

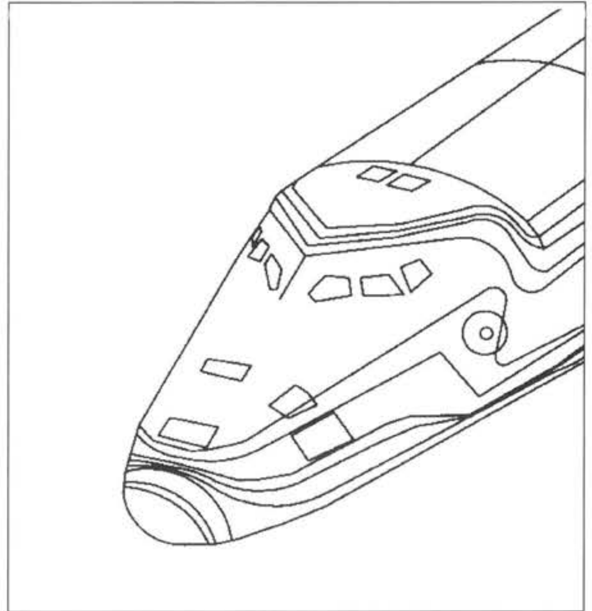


Bild 5: Gescannt mit 400 dpi.

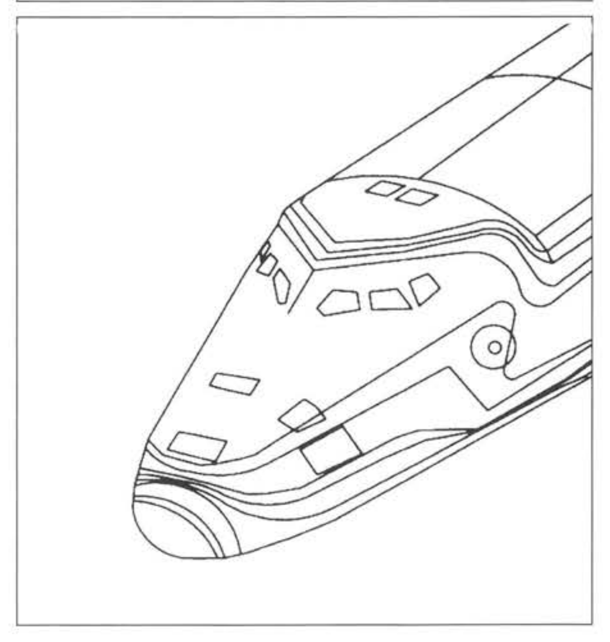


Bild 6: Gescannt mit 600 dpi.

Impressum Microsoft System Journal

Erscheinungsweise: Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats (ISSN 0933-9434). Den Heften liegt eine Diskette bei.

Herausgeber: Microsoft GmbH
Edisonstr. 1
D-8044 Unterschleißheim
Tel.: 089 / 31705-0
Telefax: 089 / 3 17 05-400
Teletex/Telex: (17) 89 83 28 MS-GmbH

Redaktion: Synergy Verlag GmbH
Redaktion *Microsoft System Journal*
Theresienstraße 40
D-8000 München 2
Tel.: 089 / 28 06 85, Telefax: 089 / 28 21 63
Günter Jürgensmeier, Hartmut Niemeier

Mitarbeiter: Marc Adler, Rainer von Ammon, Michael Bülow, Marcellus Buchheit, Greg Comeau, Daid E. Cortesi, Günter Jürgensmeier, Michael Kausch, Rainer Kreutzer, Jim Kyle, Hartmut Niemeier, Charles Petzold, Chip Rabinowitz, Michael Tischer, Jochen Witte

Manuskript-einsendungen: Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.

Typographische Konzeption: Rudolf Paulus Gorbach, D-8035 Gauting-Buchendorf

Titelbildentwurf: Hermann Menig, D-8421 Wildenberg

Anzeigen: Marketing Projekt 2000 GmbH
Gottfried-Böhm-Ring 59
D-8000 München 70
Tel.: 089 / 7 85 58 82, Telefax: 089 / 78 19 28

Druck: Alois Erdl KG, D-8223 Trostberg

Vertriebsleiter: Axel Herbschleb, Vogel Verlag und Druck KG
Postfach 6740, Max-Planck-Straße 7/9
8700 Würburg 1
Tel.: 0931 / 418-2198
Telefax: 0931 / 418-2120

Vertrieb Handelsauflage: Inland (Bahnhofsbuchhandlung):
Vereinigte Motor-Verlage GmbH & Co. KG
Leuschnerstraße 1
7000 Stuttgart 1
Tel.: 0711 / 2043-1, Telex: 7 22 036

Abbonement-Bestellungen: Vogel Verlag und Druck KG
Abonnement Service *Microsoft System Journal*
Postfach 6740
8700 Würzburg 1
Tel.: 0931 / 418-2019
Telefax: 0931 / 418-2120, Udo Kleindienst

Bezugspreise: Jahresabbonnement Inland 126,- DM (117,76 DM + 8,24 DM Umsatzsteuer), Ausland:
Österreich: 900 öS
Schweiz: 126 sfr
übriges Ausland: 132 DM
Abonnementspreise inkl. Versandkosten. Einzelheftpreise siehe Titelseite. Einzelheftpreis zzgl. Versandkosten. Schüler- und Studenten-Jahresabonnements werden mit 30% rabattiert (nur gegen Nachweis). Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.

Bankverbindungen: Für Inland und Ausland außer Österreich und Schweiz:
Abonnement:
nur auf Postgiroamt, Stuttgart (BLZ 600 100 70) 2117 17-707
Einzelheft:
nur auf Postgiroamt, Stuttgart (BLZ 600 100 70) 2385 25-705

Herstellung: Das *Microsoft System Journal* wird mit Microsoft Word 5.0 geschrieben und gestaltet. Der Ausdruck erfolgt auf einer Linotronic 300.

Vorschau

Im Heft 6/89 (November/Dezember) werden Sie unter anderem folgende Artikel finden:

MS OS/2

DDE unter dem Presentation Manager
Entwicklung eines Multithread-Programms
Bildschirmansteuerung unter MS OS/2
Aufbau und Einsatz von Dynamic Link Libraries
OS/2-Systemaufrufe von Basic aus verwenden

C

SAA-Serie, Teil 6 (Dialogboxen)
Erweiterte Techniken für Strukturen und Unions in C
C-Kurs für Umsteiger, Teil 2

Windows

Erweiterung der Windows-Dialogboxen mit neuen Steuerungsklassen

MS-DOS

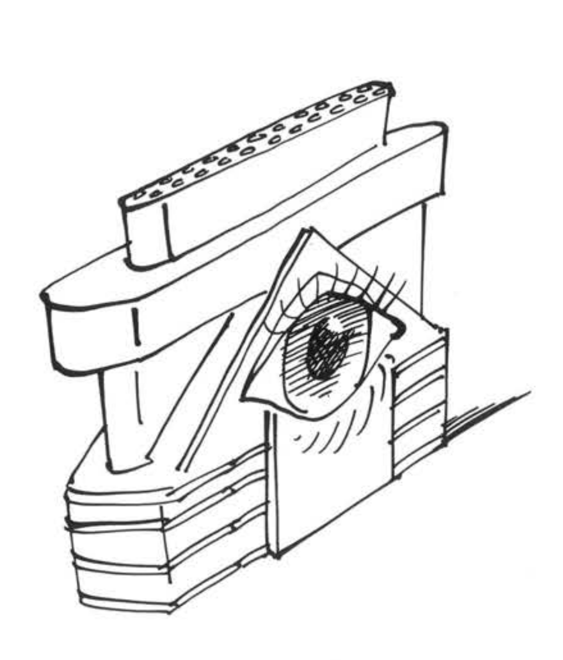
Speicherbeschränkungen umgehen mit einem Overlay-Manager
Die EXEC-Funktion von MS-DOS

Darüber hinaus werden wir über einige interessante neue Produkte von Microsoft berichten.

Inserenten

Albrecht Software	165
BKS-Software	67
ENZ	140
Fast Electronic	179
Kickstein Software	140
Marketing Projekt 2000	157
Markt & Technik Buchverlag	54/55, 180
Microsoft	9, 58, 78/79, 115
PEM	173
Schneider + Koch	95
Synergy Verlag	174, 177
Systhema Verlag	131
te-wi Verlag	2
Vogel Verlag	29, 100
Zoschke	140

Liebe Softwareknacker. Es ist völlig zwecklos, dem neuen Hardlock E-Y-E schöne Augen zu machen.







Softwareschutz braucht einen seriösen Partner.
Wer beim Schutz seiner Software kein Auge mehr zudrücken möchte, landet irgendwann bei FAST Electronic. Seit 1985 haben wir über 120.000 Hardlock-Module produziert. Und verkauft. Denn Hardlock ist kein gewöhnlicher Kopierschutz. Hardlock ist Softwareschutz durch Hardware. Transparent. Anreihbar. Und durch die einzigartige Encryption-Technique nicht zu knacken.

Hardlock E-Y-E – ein eigener Chip für den Softwareschutz.
Der Vorsprung wird größer. Das neue Hardlock E-Y-E basiert auf einem Chip, den FAST Electronic eigens zum Schutz von Software entwickelt hat. Hardlock E-Y-E ist cryptoprogrammierbar. Das heißt, es kann von Ihnen selbst programmiert werden – und von niemand anderem. So sind Sie flexibler beim Schutz von Programmoptionen. Im neuen Hardlock E-Y-E können optional bis zu 128 Byte Daten abgelegt werden. Und mit dem automatischen Einbindungsprogramm HL-Crypt schützen Sie Ihre .COM- und .EXE-Files in Sekundenschnelle. Für die individuelle Einbindung erhalten Sie ohne Aufpreis Abfrageroutinen für alle gängigen Compiler.

Testen Sie den neuen Chip.
FAST Electronic hat eine halbe Million Deutsche Mark investiert, um Hardlock noch sicherer zu machen. Aber nicht teurer. Es lohnt sich, unsere Konditionen zu kennen. Und Leistung und Preis mit anderen Systemen zu vergleichen. Sie werden sehen: Gerade bei großen Stückzahlen können wir Ihnen konkurrenzlos günstige Angebote kalkulieren. Bestellen Sie ein Test-Modul des neuen Hardlock E-Y-E unverbindlich zur Ansicht. Jetzt.



Das neue Hardlock E-Y-E: Sicherheit in Silizium.



Fast Electronic GmbH

FAST Electronic GmbH, Kaiser-Ludwig-Platz 5, 8000 München 2, Tel. (089) 53 26 53, Fax (089) 53 34 01

Einsteigen leichtgemacht!

Allgemeine Systemkommandos: **date**

MS-DOS - Arbeitsschritte

Eingabe durch den Anwender

```
A>date
Systemdatum: Di. 01.01.1980
Neues Datum (tt.mm.jj) eingeben:
30.3.89
Do. 30.03.1989
A>
```

Simulierte Antwort

Informationen zur Simulation

Das Betriebssystem hat das neue Datum übernommen und meldet sich mit dem Promptzeichen zurück. Zur Information wird zugleich der **Wochentag** angezeigt!

Sollten Sie in Ihrem Computer eine batteriegepufferte Hardware-Uhr eingebaut haben, müssen Sie diese nur einmal mittels eines mitgelieferten speziellen Programmes einstellen. Das Datum bleibt dann - während der Lebensdauer der Batterie - korrekt eingestellt.

A:

B:

F1 HILFE F2 MENÜ F3 KAPITEL F4 STICHWORT F9 ZURÜCK F10 ENDE WEITER

Auswahl der Lerneinheiten

Aufruf eines Stichwortverzeichnisses

Statuszeile des Lernprogramms

Lernen am PC:
Interaktive Lernprogramme eignen sich hervorragend, für einen ersten Einstieg in neue Themenbereiche. Sie können sich als Anfänger Grundwissen zu bestimmten Programmen aneignen und so Veränderungen in Schule und Beruf schnell gerecht werden.
Einsteigen leichtgemacht:
Durch Simulation am PC erarbeiten Sie sich schrittweise die Grundfunktionen zu den verschiedenen Programmen. Jede Lernsoftware ist in einzelne Lektionen eingeteilt. Dadurch können Sie selbst bestimmen, welche Kapitel Sie bearbeiten möchten. In kleinen Übungen am Ende eines jeden Kapitels können Sie das neu erworbene Wissen sofort umsetzen und vertiefen. Die Übungen dienen als Lernkontrolle und können beliebig wiederholt werden.

Basic Bestell-Nr. 56552
Künstliche Intelligenz Bestell-Nr. 56558
MS-DOS 3.3 Bestell-Nr. 56559
Unix Bestell-Nr. 56555
dBase III Plus Bestell-Nr. 56549
Turbo Pascal 4.0 Bestell-Nr. 56550

C Bestell-Nr. 56551
Word 4.0 Bestell-Nr. 56553
Schach Bestell-Nr. 56554
In Vorbereitung:
PC-/MS-DOS 4.0 Bestell-Nr. 56594
Multiplan 3.0 Bestell-Nr. 56593
Word 4.0, Teil 2 Bestell-Nr. 56592



Jede Lernsoftware:

DM 79,-

(sFr. 72,-/öS 790,-)
Unverbindliche Preisempfehlung

Übrigens gibt es alle Lernsoftware-Programme auf 5¼" und 3½"-Disketten. Bitte fügen Sie für die 3½"-Version bei Ihrer Bestellung am Ende der Bestell-Nummer ein »D« an.

INFO-COUPON

2511/907

Bitte senden Sie mir Ihr Gesamtverzeichnis mit 500 aktuellen Computerbüchern und Software.

Name

Straße

PLZ/Ort

Bitte ausschneiden und senden an: Markt & Technik Verlag AG, Buch- und Software-Verlag, Frau Brosien, Hans-Pinsel-Str. 2, 8013 Haar

Markt & Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

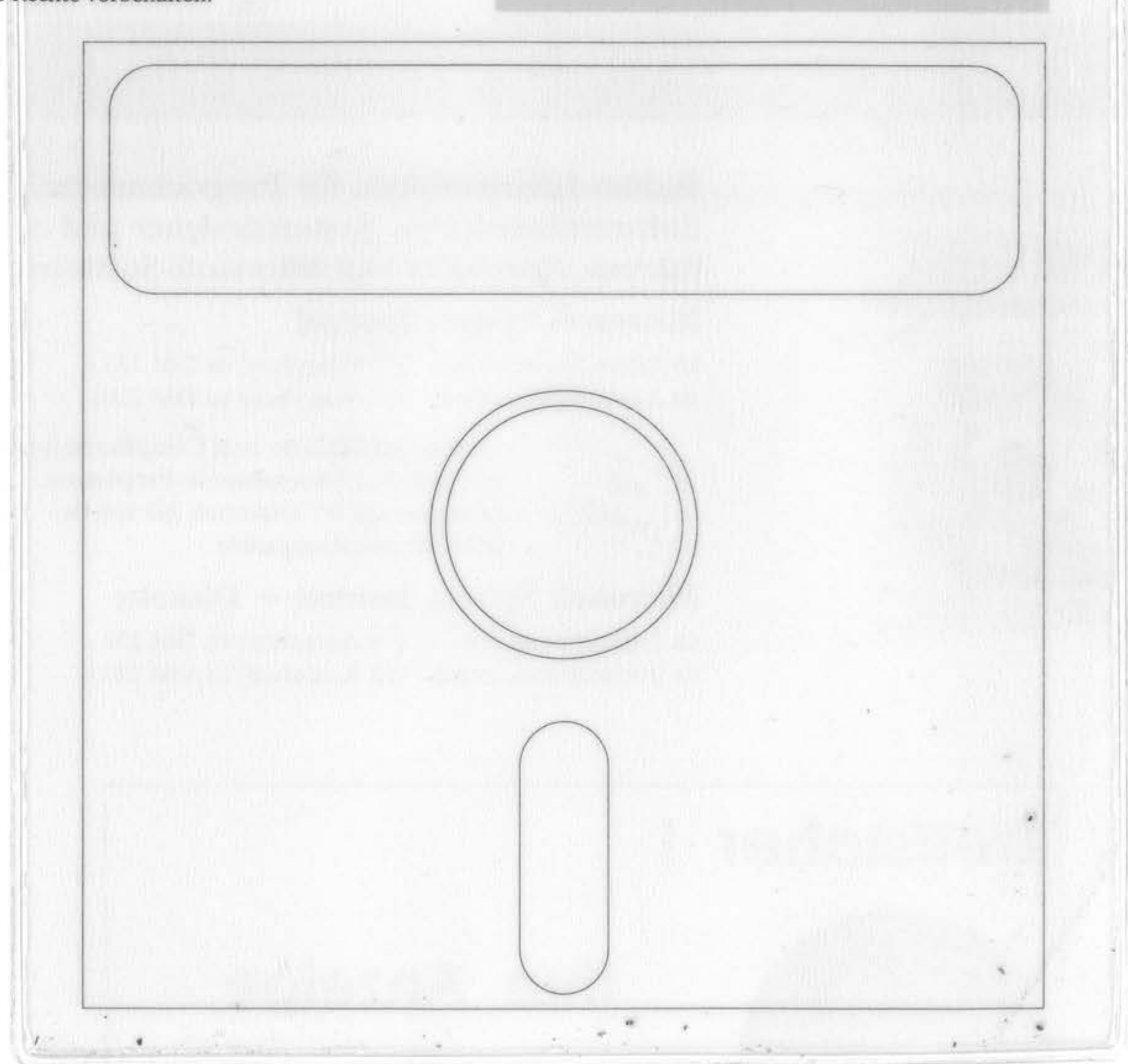


Die Diskette zum Microsoft System Journal Sept./Okt. 1989

README.EXE und READ.ME: Diese Datei und ein Programm zur Anzeige derselben.
\8259: Listings aus dem Artikel »Steuerung von Ein-/Ausgabegeräten«, S.166
\BASIC: Listings aus dem Artikel »Basic als professionelle Programmiersprache«, S.59
\OOP: Listing aus dem Artikel »Das neue Quick-Pascal«, S.68
\SAA5: Listings aus dem Artikel »Erstellung und verwaltung von SAA-Dialogboxen«, S.116
\VEC: Listings aus dem Artikel »Vektorfonts unter dem Presentation Manager«, S.6
\VMM: Listings aus dem Artikel »Eine virtuelle Speicherverwaltung in C«, S.104
\WINHELP: Listings aus dem Artikel »Hilfe-Verwaltung für Windows», S.30


```
\
|-READ.ME        README.EXE
|-\8259
| |-L1.LST       L2.LST         L3.LST
|-\BASIC
| |-GETCOLOR.BAS WORDWRAP.BAS
|-\OOP
| |-OOPDEMO.PAS
|-\SAA5
| |-DLG.C        DLGAB.C        DLGDEMO.C      DLGEDIT.C      DLG.H
|-\VEC
| |-VECTFONT     VECTFONT.C     VF00.C         VF01.C         VF02.C
| |-VF03.C       VF04.C         VF05.C         VF06.C         VF07.C
| |-VF08.C       VF09.C         VF10.C         VF11.C         VF12.C
| |-VF13.C       VF14.C         VF15.C         VECTFONT.DEF   VECTFONT.H
| |-VECTFONT.LNK VECTFONT.RC    VECTFONT.SYM
|-\VMM
| |-VM.C         VM.H
|-\WINHELP
  |-HELPMGR.C    SHAPE.C        SPLIT.C        SHAPE.DEF      DEFS.H
  |-HELPMGR.H    SHAPE.MD       SHAPE.RC       HELP.TXT
```

Verzeichnisstruktur der Diskette MSJDSK35



Sollte sich diese Diskette als defekt erweisen, schicken Sie sie bitte an folgende Adresse:

Microsoft System Journal
Vertriebsservice
Postfach 6740
D-8700 Würzburg 1

Sie erhalten dann umgehend ein neues Exemplar.

Jetzt zum Abo-Vorteilspreis bestellen!



Vorsprung sichern · aus erster Hand informieren.
MS Journal ..\etc
im Jahresabonnement
nur DM 36,–
im 2-Jahresabonnement
nur DM 64,–



Jetzt zum Abo-Vorteilspreis bestellen!



Insider-Informationen für Programmierer, Softwareentwickler, Systemdesigner und erfahrene Anwender von Microsoft-Software.

Microsoft System Journal

im Jahresabonnement (6 Ausgaben) zu DM 115,–
im 2-Jahresabonnement (12 Ausgaben) zu DM 210,–



Listings auf Diskette zum Compilieren oder zur schnellen Übernahme in Programme. Lieferbar auf 5¼"Disketten 360 KB für IBM PC und Kompatible

Microsoft System Journal + Diskette

im Jahresabonnement (6 Ausgaben) zu DM 230,–
im 2-Jahresabonnement (12 Ausgaben) zu DM 420,–